

DTD Inference for Views of XML Data

Yannis Papakonstantinou and Victor Vianu
Computer Science & Engineering, UC San Diego
{yannis,vianu}@cs.ucsd.edu

Abstract

We study the inference of Data Type Definitions (DTDs) for views of XML data, using an abstraction that focuses on document content structure. The views are defined by a query language that produces a list of documents selected from one or more input sources. The selection conditions involve vertical and horizontal navigation, thus querying explicitly the order present in input documents. We point several strong limitations in the descriptive ability of current DTDs and the need for extending them with (i) a subtyping mechanism and (ii) a more powerful specification mechanism than regular languages, such as context-free languages. With these extensions, we show that one can always infer *tight* DTDs, that precisely characterize a selection view on sources satisfying given DTDs. We also show important special cases where one can infer a tight DTD without requiring extension (ii). Finally we consider related problems such as verifying conformance of a view definition with a predefined DTD. Extensions to more powerful views that construct complex documents are also briefly discussed.

1 Introduction

Data on the World Wide Web has structure that is irregular or only partially known. This is a significant departure from the traditional database framework geared towards highly-structured data described uniformly by a rigid schema. It requires the design of appropriate languages for querying Web data, and new approaches to flexible typing and static analysis.

Recent research on semistructured data in the database community has attempted to address this challenge (see [ABS99] and the surveys [Bun97, Abi97, Suc]). The emergence of the Extended Markup Language (XML) as the likely future standard for representing data on the Web has confirmed the central role of semistructured data but has also redefined some of the ground rules. Perhaps the most important is that XML marks the “return of the schema” (albeit loose and flexible) in semistructured data, in the form of its

Data Type Definitions¹ (DTDs). This is significant, because schematic information is essential at all levels of database design, implementation, and usage.

DTDs describe the structure of the objects (or “elements”) participating in an XML document. A DTD specifies, for each type of object, the allowed sequences of types of its subobjects (see Figure 1 for an example DTD). Additionally, DTDs may specify information such as attributes or special content for each type. DTDs can have multiple uses in creating views and querying XML data. QBE-style query interfaces [MP, B⁺99] may use DTDs to display the “schema” of a view and allow users to navigate it. DTDs may help in the design of the storage structures. Mediators, which create integrated views by selecting and restructuring source data, use DTDs to optimize the queries that they send to the sources [PV, PV99a]. Semistructured databases use DTDs to semantically optimize their query plans [FS98]. Finally DTDs may guide the production of style sheets, such as XSL scripts [CD], that display XML documents as browser-compatible HTML documents [Si, Bi].

It is clear that DTDs will be particularly useful. Mediators and databases that create views of XML data will have to export the views’ DTDs. However, creating a view DTD “manually” by delving into the details of the source DTDs is error-prone and may become the bottleneck of the (integrated) view development. This paper studies the *automatic* inference of view DTDs from source DTDs.

Formal framework We use an abstraction of XML documents and DTDs that focuses on document *structure*. XML documents are modeled as ordered trees with labeled nodes. Nodes correspond to XML elements and their labels provide the type names of the elements. The children of a node are totally ordered. A DTD is modeled as a *labeled tree definition (ltd)*, that associates with each type name a language on the alphabet of type names. Although DTDs use only regular languages, we also consider ltds that use more powerful languages, such as context-free.

To study DTD inference for views we introduce a view definition language that queries labeled ordered

¹The recent XML-Data and DCD [LJM⁺] standards also provide “loose” schemas for XML documents.

trees. The language allows conditions on the order of elements in the input and also controls the order of elements in the output. Queries extract variable bindings from the input using a tree pattern involving *regular expressions* to navigate both vertically and horizontally. Horizontal regular expressions provide a powerful way to query the order of the nodes. They are encountered in some semistructured query languages [CDSS98] and seamlessly couple with the more commonly used regular path expressions for vertical navigation [Suc, Abi97, Bun97, CM90, MW95, dBV93, AQM⁺97, BDHS96, FFLS98, FLS98, AM98, DFF⁺, KS95, AV97]. In a sense, our language enhances the horizontal navigation functionality of the XPointer standard [MD] and incorporates it into a semistructured query language. The variable bindings extracted by the tree pattern are used to construct the answer. We focus on *selection queries*, which extract from the input the list of subtrees to which one of the variables in the tree pattern binds. We only consider queries whose condition ranges over one source/tree only. The generalization to multiple sources is straightforward, since these can be viewed as one source obtained by concatenating the multiple sources (see Examples 2.12 and 2.16). Loto-ql provides the formal basis for the query and view definition language XMAS [VLP00], which is implemented and used within the MIX project [B⁺99]. A DTD inference algorithm following the steps outlined in this paper has been implemented for XMAS.²

Results Given a source ltd and a view definition, we study the problem of constructing a *tight* ltd for the view, i.e., an ltd that precisely characterizes the type structure of trees in the view. Our quest for the tight ltd quickly highlights two severe limitations of current DTDs in XML. The first is that DTDs lack a subtyping mechanism; the repercussions are numerous. For example, there is no tight DTD for the set of documents from two sources, each with its own DTD, or for other very simple views. We overcome these limitations by enhancing ltds (our abstraction of DTDs) with a simple subtyping mechanism, called *specialization*, which is in the spirit of union types. Despite their simplicity, specialized ltds encompass the expressive power of formalisms such as dataguides [NUWC97, GW97] and graph schemas [BDFS97] and they are of equal power with the formalism proposed in [BM99, CDSS98]. Interestingly, specialized ltds turn out to specify precisely the regular tree languages

of finite unranked³ trees, see [BKMW].

The second limitation is that DTDs only use regular languages; this is generally insufficient for describing views. We can overcome this problem by allowing context-free languages in ltd specifications. The main result of the paper is that the two proposed extensions suffice for selection queries. We provide an algorithm to construct, from every source ltd and view defined by a selection query, a tight ltd for the view that uses context-free languages and specialization. The construction uses an array of classical techniques from language theory.

The above algorithm provides a solution to the tight ltd inference problem for selection queries, but comes at the cost of using the extended ltds. Some applications may choose to provide ltds that are simply *sound* for the view, i.e. ltds that are satisfied by all trees in the view but may also allow trees that are not in the view. If one prefers to give up tightness in return for using regular ltds (corresponding to existing DTDs), there is good and bad news. The bad news is that it is undecidable if a selection view has a tightest regular ltd. The good news comes in several flavors:

- It can be checked whether a selection view conforms to a predefined regular ltd; this makes use of the tight specialized context-free ltd we can infer. Note that conformance is important for applications that expect their input, which will be the XML result of a query/view, to satisfy predefined DTDs.
- Tight specialized regular ltds can be inferred for selection views in several special cases of practical interest. For example, one case is when the source ltd is *stratified*, i.e. no type uses itself in the ltd directly or indirectly. Another is when the view is defined by a query involving only non-recursive vertical navigation. If specialization cannot be used, one can still infer in these cases a tightest regular ltd for the views.

Most query languages for semi-structured data provide mechanisms for constructing complex XML documents as answers to queries. For such views, our inference algorithm can be extended to produce a sound ltd, but no longer tight (due to space limitations, the extension is presented in the appendix). Intuitively, tightness is lost due to dependencies among variables that cannot be captured by specialized context-free ltds. It remains open if this can be remedied with a more powerful type system.

Related work Type checking and type inference are well-studied problems in functional programming [Mit90, Mit96]. The problem we study is similar in

²The code is available at <http://www.db.ucsd.edu/Projects/MIX/SchemaInference/>. Note that XMAS expresses order conditions using precedence relationships of the form “the object *X* precedes the object *Y*.” This form of order conditions simplifies to some extent the DTD inference algorithm.

³In an unranked tree, the number of children of each node is unrestricted, and the children are ordered.

flavor to type inference. However, despite the superficial similarity, there do not appear to be substantive technical connections between DTD inference (which involves language-theoretic machinery) and classical type inference in programming languages.

A comprehensive presentation of recent research in semi-structured data can be found in [ABS99]. Apart from DTDs, notable approaches to specifying schematic information in semi-structured data include representative objects and dataguides [NUWC97, GW97] and graph schemas [BDFS97]. The problems they address are orthogonal to ours. Inference of schemas for views is not considered. The “patterns” in [CDSS98] are a form of dataguide. A limited form of inference can be accomplished by inferring the patterns that view variables may bind to.

DTDs are used in [MZ98] for schema matching, which, in turn, is used for data conversion. Specialized ltds can also support these activities. In [NAM98], another approach to inferring a schema from graph data is proposed, which results in a classification of the nodes in a class hierarchy.

Like our language, the languages of [AM98, CDSS98] control the order of elements in the output. The language of [AM98] also places conditions on the order of elements in the input.

Type checking and inference for views defined by selection queries on ordered graph data are considered in [MS99]. The selection queries return *sets* of objects selected from the input using vertical navigation only. The input data is assumed to satisfy a given DTD. Types of the output consist of label assignments to variables of the query. Since the objects in the result are not ordered, the output type does not describe the allowed *sequences* of labels, unlike the types we consider. Also, [MS99] does not infer tight type descriptions and they do not consider the specialization of a type as a result of the conditions that are imposed on it. Type checking in [MS99] consists of verifying if a given label assignment to the query variables is possible for some input graph satisfying the input DTD. Type inference consists of finding all satisfiable label assignments. The results focus on the complexity of type checking and inference.

Powerful selection queries on trees are studied using Attribute Grammars in [NdB98, Nev99] and Query Automata in [NS99]. The results concern expressiveness of the languages and complexity of static analysis questions such as emptiness, equivalence, and circularity.

A discussion of problems raised by schema inference in views of semistructured data is presented in [PV99b]. The notions of sound and tight DTD are defined, and the need for specialized DTDs is illustrated. An algorithm is presented for inferring the DTD of selection views with no horizontal navigation

and no recursive path expressions.

The tight ltds inferred by our algorithm for selection queries provide, as a side-effect, a test of conformance to a predefined ltd. As discussed earlier, for more complex queries with constructed answers our algorithm only produces sound (but no longer tight) ltds. Sound ltds can only provide a sufficient test of conformance to a predefined ltd. A direct approach to testing conformance for a broad class of views is developed in [MSV], using *inverse type inference*.

The paper is organized as follows. The next section provides a warm-up to the main development. It introduces the main concepts and notation, considers several basic properties of ltds used throughout the paper, and motivates the extensions of ltds with specialization and context-free languages. Section 3 introduces the view definition language. Section 4 contains the main results on ltd inference, and discusses some interesting special cases. An appendix contains details of some constructions, examples and inference of sound ltds for loto-ql queries with constructed answers.

2 Warm-Up

In this section we present the formal framework of the paper, and motivate the extension of DTDs with a subtyping mechanism and with context-free grammars.

We assume familiarity with basic notions of language theory, including (nondeterministic) finite-state automata ((n)fsa), context-free grammar (CFG) and language (CFL), homomorphism, substitution, and sequential transducer (e.g., see [HU79, Gin66]). We will use basic facts such as closure of regular and context-free languages under homomorphism, inverse homomorphism, intersection with regular languages, substitution with languages of the same kind, and sequential transducers.

We also use results on regular tree languages and tree automata. Regular languages of finite binary trees are surveyed in [GS97]. The unranked case is discussed in [BKMW]. Regular tree languages have similar closure properties to regular string languages, in both the ranked and unranked cases. Emptiness of the language accepted by a tree automaton is PTIME-complete, and inclusion is EXPTIME-complete. A brief description of tree automata is provided in Appendix A.

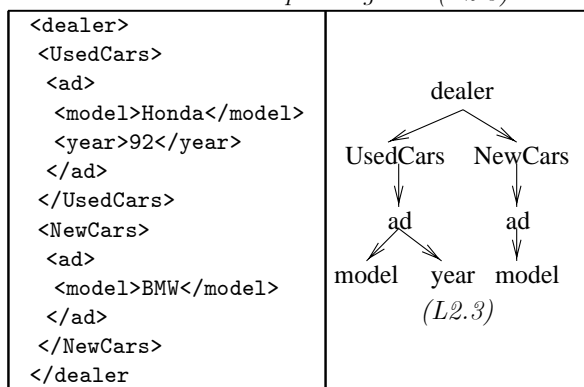
Labeled ordered tree objects A labeled ordered tree object (loto) is our abstraction of an XML document. Each node represents an XML element and is labeled by the element’s name (type). The list of children of a node represents the sequence of elements

that make up the content of the node, labeled by their name.

Definition 2.1 A labeled ordered tree object (loto) over alphabet⁴ Σ is a finite tree such that each node has an associated label in Σ and the set of children of a given node is totally ordered⁵.

Given a loto t and a node n of t , we denote the label of n by $\lambda(n)$. Thus, if the sequence of children of n is $n_1 \dots n_k$ then $\lambda(n_1) \dots \lambda(n_k)$ is a word in Σ^* . If n is a node in a given loto, we denote by $tree(n)$ the subtree of the loto rooted at n .

Example 2.2 Consider the following “dealer” XML document and the corresponding loto (L2.3).



Note that the loto retains the element name structure of the XML document. However the string content of the document is discarded since it plays no role in the inference problem.

Loto type definitions A loto type definition (ltd) is our abstraction of DTDs. It retains the element content information provided by DTDs [BPSM]. In particular, a DTD specifies, using regular expressions, the sequences of element names that are allowed as content of elements with a given name. The DTD also specifies the type of the root. These are formalized as follows.

Definition 2.4 A loto type definition (ltd) over alphabet Σ consists of a root type in Σ and a mapping associating to each $a \in \Sigma$ a language over Σ .

By slight abuse of notation, if d is an ltd over Σ , we denote by $d(a)$ the language over Σ associated with a . We also denote the type of the root by $d(\text{root})$.

The languages provided by an ltd can be specified by various means, and for simplicity we often blur the distinction between a language and its specification. If the languages are regular, they can be represented by regular expressions over Σ . We call such an ltd *regular*. Similarly, an ltd whose languages are context-free (and

```

<DOCTYPE dealer [
  <!ELEMENT dealer (UsedCars,NewCars)>
  <!ELEMENT UsedCars (ad*)>
  <!ELEMENT NewCars (ad*)>
  <!ELEMENT ad ((model,year)|model)>
]>

```

Figure 1: An XML DTD corresponding to Example 2.5

specified by a CFG or other means) is a *context-free ltd*. An ltd is assumed by default to be regular.

The function of a loto type definition is to specify a set of valid lotos, analogously to the way a DTD specifies a set of XML documents conforming to it. A loto t satisfies an ltd d over Σ if the root has type $d(\text{root})$ and for every node n of t with children $n_1 \dots n_k$, the word $\lambda(n_1) \dots \lambda(n_k)$ is in $d(\lambda(n))$. The set of lotos satisfying an ltd d is denoted by $T(d)$.

Example 2.5 The loto (L2.3) satisfies the ltd below. For readability, in ltd examples we denote concatenation by comma. The examples specify the root type and the languages associated to each type name. We omit specifying a language if it is $\{\epsilon\}$ – e.g., for model and year below.

```

root : dealer;
dealer : (UsedCars, NewCars);
UsedCars : ad*;
NewCars : ad*;
ad : (model, year) + model;

```

(LTD2.6)

Figure 1 provides a corresponding DTD.

Note that, in order for an ltd to be satisfiable by some loto, the ltd has to provide “exit rules”, i.e. some of the $d(a)$ must contain ϵ . In the example, $d(\text{model}) = d(\text{year}) = \{\epsilon\}$, and $d(\text{UsedCars}), d(\text{NewCars})$ contain ϵ . Clearly, a regular ltd may be viewed as an extended CFG (in an extended CFG, productions have regular expressions on the right-hand side, with the obvious meaning.) The lotos satisfying a given ltd are the derivations in the corresponding extended CFG.

We say that two ltds d and d' are *equivalent* if $T(d) = T(d')$. An ltd d is *tighter* than an ltd d' if $T(d) \subseteq T(d')$. Checking either property turns out to be PSPACE-complete (the lower bound follows from the fact that regular expression containment and equivalence are PSPACE-complete [GJ79]).

We will consider throughout the paper sets of lotos constructed by various means from other sets of lotos satisfying given ltds. For example, views of lotos satisfying a given ltd generate such new sets. This leads naturally to the question of which sets of lotos can be described by ltds. There are two orthogonal requirements in order for a set T of lotos to be specifiable by an ltd:

⁴We only consider finite alphabets.

⁵Thus, each node has a list of children; if there are no children, the list is ϵ .

- (i) For each $a \in \Sigma$, let L_a be the language consisting of all words $\lambda(n_1) \dots \lambda(n_k)$ for which $n_1 \dots n_k$ is the list of children of some node n with label a in some loto in T . Then L_a has to be regular (or of appropriate kind for non-regular ltds).
- (ii) T must be closed under substitution of subtrees with the same root type. More precisely, if t is in T , n is a node in t , and n' is a node in some loto in T such that $\lambda(n) = \lambda(n')$, then the loto obtained from t by replacing the subtree $tree(n)$ by $tree(n')$ is also in T .

We will refer informally to property (ii) as closure under subtree substitution.

Example 2.7 As an illustration of (ii), consider the singleton set $T = \{(L2.3)\}$ (see Example 2.2). Clearly, T violates (ii); thus, it cannot be specified by any ltd. Intuitively, the problem is that no ltd can specify one structure for the ad in UsedCars and another for the ad in NewCars. Note that T trivially satisfies (i), since the language associated to each element name is finite and therefore regular.

A set T of lotos may satisfy (ii) and violate (i).

Example 2.8 Consider the set of lotos described by the following ltd.

(LTD2.9) $root : section;$
 $section : intro, section^*, conclusion;$

Now consider a query that collects all intro and conclusion nodes of a given loto and groups them under a root named result, in exactly the same order in which they appear in the input. It is easy to see that L_{result} is not a regular language but it is context-free.

We will say that an ltd d is *tight* for T if $T = T(d)$. It is easy to show the following:

Lemma 2.10 A set T of lotos has a tight ltd iff it satisfies (i) and (ii) above.

If T does not have a tight ltd, it is still of interest to find an “approximate” description of T . A dtd d is *sound* for T iff $T \subseteq T(d)$. In general, there are many ltds that are sound for given T ; among the candidates, the best would be the tightest sound ltd, if such exists. The following characterizes the sets T for which tightest sound ltds exist.

Lemma 2.11 A set T of lotos over alphabet Σ has a tightest sound regular ltd iff each language L_a (defined in (i) above) is regular for all $a \in \Sigma$.

The tightest sound ltd for T satisfying the property in the lemma is simply the ltd d such that $d(a) = L_a$. Consequently, a set T of lotos cannot have several incomparable sound ltds that are minimal with respect to tightness. In other words, either there exists a unique tightest ltd, or for every sound ltd there exists a strictly tighter sound ltd. For example, given the following sound ltd for the view of Example 2.8

$root : result; result : (intro + conclusion)^*$

we can come up with the strictly tighter ltd

$root : result;$

$result : \epsilon + intro, (intro + conclusion)^*, conclusion$

which can be tightened ad infinitum.

Specialized loto type definitions Closure under subtree substitution seriously limits the specification power of ltds in many practical cases. Example 2.7 showed a single loto that cannot be described by a tight ltd. Similarly, union of sets of lotos specified by two ltds do not generally have a tight ltd, as illustrated next.

Example 2.12 Consider two sources exporting lotos conforming to the following ltds.

$root : UsedCars;$ $UsedCars : ad^*;$ $ad : model, year;$	$root : NewCars;$ $NewCars : ad^*;$ $ad : model;$
---	---

Now consider a new source obtained by the concatenation of the two sources under a loto “all”. The tightest ltd for the new source is listed below; but it is not tight.

$root : all; all : UsedCars, NewCars;$

$UsedCars : ad^*; NewCars : ad^*$

$ad : (model, year) + model;$

A tight specialized ltd for the concatenation of the two sources is shown in Example 2.12.

The following further illustrates the shortcomings of ltds in describing views.

Example 2.13 Consider the following source ltd and a view that collects all dealers that sell at least one used vehicle and groups them under a “used-dealers” node. The tightest ltd for the view is identical with the source ltd — modulo renaming dealers to UsedDealers. Thus, the ltd cannot capture the fact that at least one “used dealer” ad must be for a used car.

$root : dealers; dealers : dealer^*;$

$dealer : ad^*; ad : UsedAd + NewAd;$

The shortcomings illustrated above have a common source. They are due to the inability of ltds to carry typing information across multiple levels of the trees (lotos) they describe. This is reflected in the closure under subtree substitution of sets of lotos with tight ltds. Intuitively, overcoming this limitation requires the ability to define special cases of a given type. Indeed, it turns out that this simple idea allows to overcome the limitations of ltds mentioned above. We next define specialized ltds.

Definition 2.14 A specialized ltd for alphabet Σ is a 4-tuple $\langle \Sigma, \Sigma', d, \mu \rangle$ where:

(1) Σ, Σ' are finite alphabets;

(2) d is an ltd over Σ' ; and,

(3) μ is a mapping from Σ' to Σ .

Intuitively, Σ' provides for some $a \in \Sigma$, a set of specializations of a , namely those $a' \in \Sigma'$ for which $\mu(a') = a$. Note that μ induces a homomorphism on words over Σ' , and also on lotos over Σ' (yielding lotos over Σ).

We also denote by μ the induced homomorphisms. Specialized ltds are denoted by bold letters $\mathbf{d}, \mathbf{e}, \mathbf{f}$, etc.

Let $\mathbf{d} = \langle \Sigma, \Sigma', d, \mu \rangle$ be a specialized ltd. A loto t over Σ satisfies \mathbf{d} if $t \in \mu(T(d))$.

Example 2.15 *The following specialized ltd is tight for the singleton set $\{(L2.3)\}$. For readability, the set Σ' of Definition 2.14 is omitted (Σ' implicitly consists of all symbols on the left hand side of the mappings) and the mapping μ from Σ' to Σ is implicit; symbols of the form a^b map to a and symbols without superscripts map to themselves.*

root : dealer;
dealer : (UsedCars, NewCars);
UsedCars : ad^u ; *NewCars* : ad^n ;
ad^u : model, year; *adⁿ* : model;

Example 2.16 *Following is he tight specialized ltds for the source obtained by concatenating the two sources in Example 2.12.*

root : all;
all : (UsedCars, NewCars);
UsedCars : $(ad^u)^*$; *NewCars* : $(ad^n)^*$;
ad^u : model, year;
adⁿ : model;

Similarly, a specialized ltd can be obtained for the set described in Example 2.13.

Interestingly, specialized ltds turn out to have the same descriptive power as the regular tree automata over unranked finite trees, and so specify precisely the regular languages of finite, unranked trees. Indeed, the following is easily shown:

Lemma 2.17 *A set of lotos equals $T(\mathbf{d})$ for some specialized ltd \mathbf{d} iff it is a regular tree language.*

It follows that the results on regular tree languages, such as decidability of emptiness, inclusion, closure under complementation, etc, also apply to specialized ltds. Checking if a loto satisfies a fixed specialized ltd can be done in $O(n^3)$, using standard parsing techniques.

3 A query language for lotos

We present a query language for lotos, called loto-ql, similar in spirit to several query languages recently proposed for XML and the Web. Like [CDSS98, AM98], our language handles order explicitly. We focus here on *selection* loto-ql queries (full loto-ql queries are defined in Section D).

A selection loto-ql query is of the form

select X *where* *body*

where *body* is a pattern providing bindings of X and the other variables to subtrees of the input loto. The

pattern is in the shape of a tree and uses regular expressions for navigating both vertically and horizontally in the input loto (thus, the query language makes use of the order on children available in lotos). The answer is a loto consisting of the list of the subtrees to which X binds, under a new default root. The subtrees are listed in the order in which they occur in a depth-first, left-to-right traversal of the input loto.

We now define the patterns used in loto-ql queries. A pattern over alphabet Σ is a tree with labeled nodes and edges. The root has outdegree one. A node label is an expression $p_0.X_1.p_1.X_2.p_2 \dots X_k.p_k$, $k \geq 0$, where the X_i are variables and the p_i are regular expressions over Σ . Furthermore, $\epsilon \notin p_i, 0 \leq i < k$. All nodes have labels, restricted as follows: the root is labeled by a symbol in Σ , and internal nodes must contain at least one variable. Each variable occurs only once in the body. For simplicity, a regular expression equal to ϵ is omitted.

An edge outgoing from an internal node n is labeled by a pair $\langle X, p \rangle$ where X is a variable occurring in the label of n and p is a regular expression over Σ . The edge outgoing from the root is labeled simply by a regular expression p over Σ . Intuitively, an edge label describes vertical navigation in the input loto. For each node reached by vertical navigation, the node label describes horizontal navigation in the list of its children.

Formally, let B be the body of a loto-ql query over Σ and t be a loto over Σ , such that the roots of B and t have the same label. Let $Var(B)$ be the set of variables in B . A *binding* is a mapping β from $Var(B)$ to the nodes of t such that $\beta(\text{root}(B)) = \text{root}(t)$ and for each edge in B labeled $\langle X, p \rangle$ with target node labeled $p_0.X_1.p_1.X_2.p_2 \dots X_k.p_k$ there exists a path with nodes $x_0 \dots x_n$ in t where:

- $x_0 = \beta(X)$,
- $\lambda(x_1) \dots \lambda(x_n) \in p$,
- x_n has children $y_1^0 \dots y_{i_0}^0 \dots y_1^k \dots y_{i_k}^k$ where $\lambda(y_1^j) \dots \lambda(y_{i_j}^j) \in p_j, 0 \leq j \leq k$, and $\beta(X_j) = y_{i_j-1}^{j-1}, 1 \leq j \leq k$.

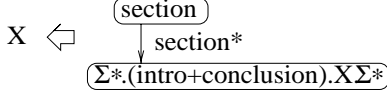
A binding must satisfy the analogous condition for the special case of the edge outgoing from the root.

Thus, an edge labeled by $\langle X, p \rangle$ leads to the nodes x_n in the input loto reachable from the node X by a path whose node labels spell a word in p . The label $p_0.X_1.p_1.X_2.p_2 \dots X_k.p_k$ of the target node of the edge provides a pattern matched against the sequence of children of x_n .

Note that the bindings for a given variable are naturally ordered by a depth-first left-to-right traversal of the input loto.

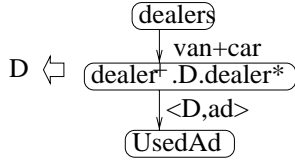
Example 3.1 *Consider the source described by the ltd of Example 2.8. The following query collects all*

intro and conclusion nodes and groups them under a root named result, in exactly the same order in which they appear in the input.



Example 3.2 Next consider the source described by the following ltd and the query that retrieves all van or car dealers that sell at least one used vehicle.

root : dealers
dealers : truck, van, RV, car;
truck : dealer*; van : dealer*;
RV : dealer*; car : dealer*;
dealer : ad*; ad : UsedAd + NewAd;



Beyond the immediate focus of the paper on ltd inference, we believe that loto-ql can serve as a useful vehicle for investigating aspects of handling order in queries for semistructured data.

4 Inferring ltds for selection views

In this section we present our results on inference of ltds for views defined by selection loto-ql queries. As discussed in the previous section, regular ltds are insufficient for describing views defined by selection loto-ql queries. Moreover, it is not even possible to *check* if the view can be specified by a regular ltd, or whether it can be approximated by a tightest regular ltd:

Theorem 4.1 *It is undecidable, given a selection loto-ql query q and an ltd d , whether $q(T(d))$ has a tight regular ltd, or whether it has a tightest regular ltd.*

The proof uses Lemma 2.10 and the undecidability of whether a CFG defines a regular language [HU79].

For the purpose of enhancing the specification power of regular ltds, we suggested extending them in two ways:

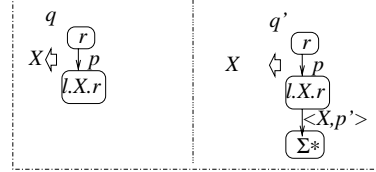
- (i) adding a *specialization* mechanism, and
- (ii) allowing specifications of content more powerful than regular expressions, such as CFGs.

The main result of this section states that one can construct tight specialized context-free ltds for all views defined by selection loto-ql queries, whose in-

put lotos satisfy a given regular ltd⁶. The result requires developing some technical machinery; most of the section is devoted to this development.

Basic Concepts and Algorithms

First we describe the ltd inference algorithm for two queries, q and q' , that illustrate several key aspects of the algorithm. Then we briefly outline the algorithm for arbitrary selection queries.



In query q , the pattern of the body says that the parent of $root(X)$ in the input loto is reachable from the root of the loto by a path spelling a word in p . The subtree X is extracted from the content of the parent by matching the expression $l.X.r$, so that $root(X)$ is labeled by the last letter of a word in l and it is followed by a suffix in r . Query q' is the same as q , except that there must also exist a downward path in p' originating at $root(X)$.

Satisfiability and validity Before describing how ltds are inferred for q and q' , we make a brief digression to consider a technical problem that arises in all cases. This relates to the condition requiring the existence of a downward path from nodes of a given type. It occurs as an explicit condition in the body of q' , but also arises in a more subtle form in q .

Consider an ltd d and a regular expression p over Σ . Let a be in Σ , and consider the question of whether there is a path in p from nodes of type a in lotos satisfying d . There are three possibilities:

- there is a path in p originating at nodes of type a in *some* lotos satisfying d (and then we say that p is *satisfiable* at a),
- there is *never* a path in p originating at a node of type a in a loto satisfying d (and we say that p is *unsatisfiable* at a)
- there is *always* a path in p originating at a node of type a in any loto satisfying d (and we say that p is *valid* at a).

In the inference algorithm, we will need to check whether a path p is satisfiable or valid at some type a . We can show the following useful fact:

Lemma 4.2 *Given a regular ltd d , a regular path expression p over Σ and $a \in \Sigma$: (i) it can be checked in PTIME whether p is (un)satisfiable at a ; (ii) it can be checked in EXPTIME whether p is valid at a .*

⁶The inference algorithm also works for inputs described by *specialized* regular ltds, such as those obtained by concatenating multiple sources, each with its own regular ltd (as in Example 2.12).

Satisfiability and validity can be reduced to questions involving regular tree languages. To see this, note that the set of lotos rooted at a and satisfying d forms a regular tree language R_d , for which a non-deterministic top-down tree automaton (for unranked trees) can be constructed from d in PTIME. Similarly, the set of lotos rooted at a for which there exists a path in p starting from the root is also a regular tree language R_p , for which an automaton can be constructed from p in PTIME. Thus, satisfiability of p at a is reduced to checking non-emptiness of $R_d \cap R_p$ (which can be done in PTIME) and validity is reduced to checking that $R_d \subseteq R_p$ (which takes EXPTIME).

Type Tightening Next, suppose a path p is satisfiable but not valid at type a defined by an ltd d . This means that a proper subset of the lotos satisfying d and having roots of type a have a path in p starting at the root. For the inference algorithm, we will need to precisely describe this set of lotos. We can achieve this by constructing a specialized ltd that provides a *tightening* of d in which p becomes valid at a . To see that this is possible, note that the desired tightened set of lotos equals $R_d \cap R_p$, which is a regular tree language. By Lemma 2.17, there exists a specialized ltd specifying it. We outline its construction in Appendix B, and denote the resulting specialized ltd by $tighten(a, d, p)$.

Vertical and horizontal navigation We now return to the example queries q and q' , which involve simple vertical and horizontal navigation. Consider first query q . Suppose q and the input ltd d are over alphabet Σ . The ltd d_q for the view defined by q is the following. The type of the root is a default *root*. Suppose for simplicity that $root \notin \Sigma$ (the other case is handled using specialization). For $a \in \Sigma$, $d_q(a) = d(a)$. The language $d_q(root)$ is a language over Σ , denoted L_X and defined by the following extended CFG G . Let f_p be an fsa over Σ accepting p ; its state transition function is δ . For each state h in f_p , let p_h be the regular language accepted by f_p with start state h . The nonterminals of G are the pairs $\langle h, a \rangle$ where h is a state of f_p and $a \in \Sigma \cup \{root\}$. The start symbol is $\langle s, root \rangle$ where s is the start state of f_p . The set of terminal symbols of G is Σ . We describe the productions of G in two steps. First, consider a nonterminal $\langle h, a \rangle$, where h is a non-accepting state of f_p . For each such $\langle h, a \rangle$, G contains the following production:

$$\langle h, a \rangle \rightarrow \sigma_h(d(a))$$

where σ_h is a substitution defined as follows. For $b \in \Sigma$ and $h' = \delta(h, b)$:

- $\sigma_h(b) = \{\langle h', b \rangle\}$ if $p_{h'}$ is valid at b ,
- $\sigma_h(b) = \{\epsilon, \langle h', b \rangle\}$ if $p_{h'}$ is satisfiable but not valid at b , and

- $\sigma_h(b) = \{\epsilon\}$ if $p_{h'}$ is unsatisfiable at b .

Note that in the case considered above where h is non-accepting, the productions only have to account for vertical navigation along p , since no horizontal matching occurs. Now suppose h is an accepting state of f_p . Things are more complicated, because the production has to also account for horizontal matching. This can be done by applying a sequential transducer t_h to $d(a)$, which simultaneously applies the substitution σ_h and performs the matching of $l.X.r$ against $d(a)$. The transducer works as follows. Given an input word w , it outputs $\sigma_h(b)$ for each symbol b of w which does *not* match $l.X.r$. If a match occurs, t_h outputs $b\sigma_h(b)$. To detect a match, t_h identifies the last letter of each prefix of w which is in l and for which the remainder suffix is in r (the nondeterminism arises in guessing whether or not the suffix from a current position is in r , and acceptance requires checking that all guesses along the way were correct). Since $d(a)$ is regular and regular languages are closed under sequential transducers [Gin66], $t_h(d(a))$ is regular. The grammar G contains, for each $\langle h, a \rangle$ where h is accepting, the production

$$\langle h, a \rangle \rightarrow t_h(d(a)).$$

The grammar G can be effectively constructed from d and q in EXPTIME (polynomial in d and exponential in q). The construction is illustrated in Example C.1 in the appendix. In summary, the view defined by q on inputs satisfying d has a tight context-free ltd constructible in EXPTIME.

Next, consider the query q' . It is very similar to q , with the difference that the nodes of type $a \in \Sigma$ to which X binds must be restricted to ensure the existence of the downstream path in p' . This requires the ltd tightening outlined earlier, and highlights the need for specialization. The language L_X constructed for q must be modified using the specialized ltds $tighten(a, p', d)$. More precisely, let \mathbf{d}' be the specialized ltd defined as follows. \mathbf{d}' defines the content of internal nodes by $tighten(a, p', d)$, $a \in \Sigma$ (observe that these specialized ltds agree on the types they share). The content of the root is defined by the language $\sigma(L_X)$ where σ is the substitution defined next. We denote by a' the root type in $tighten(a, p', d)$:

- $\sigma(a) = \{a'\}$ if p' is valid at a ,
- $\sigma(a) = \{\epsilon, a'\}$ if p' is satisfiable but not valid at a , and
- $\sigma(a) = \{\epsilon\}$ if p' is unsatisfiable at a .

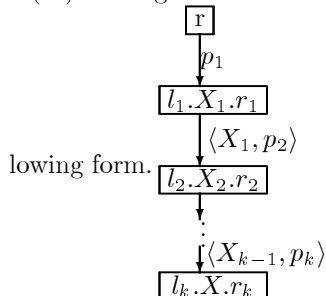
It is easy to verify that \mathbf{d}' is a tight specialized context-free ltd for $q'(T(d))$. The construction is illustrated in Example C.2 in the appendix.

Extension to Full Selection Queries

The above discussion contains in a nutshell the basic ingredients of our ltd inference algorithm for selection views. We now outline the main steps in the full algorithm, omitting many details.

Satisfiability and Tightening Revisited We have seen in the previous discussion that one can test if a regular path expression p is satisfiable or valid at $a \in \Sigma$, given a regular ltd d (see Lemma 4.2). We have also seen how a specialized regular ltd can be constructed to “tighten” a type definition in order to ensure the existence of a downward path in p from nodes of that type. Both results can be extended to arbitrary patterns: for each type a , regular ltd d , and loto-ql query body $B(\vec{X})$, all over Σ , one can test in PTIME whether $B(\vec{X})$ is satisfiable at a , and in EXPTIME whether $B(\vec{X})$ is valid at a (where satisfiability and validity of a pattern at a are the obvious extensions of the notions we defined earlier for regular path expressions). Furthermore, one can construct a specialized regular ltd $tighten(a, d, B(\vec{X}))$, whose size is polynomial in d but exponential in $B(\vec{X})$ and is satisfied precisely by the lotos $t \in T(d)$ with root type a , in which there exists a binding of $B(\vec{X})$. As a useful side effect, the construction provides, for each variable X in the body, a subset Σ_X of the types of $tighten(a, d, B(\vec{X}))$ to which X may bind.

Construction of the specialized CF ltd Consider a selection loto-ql query q over Σ , of the form *select* X where $B(\vec{X})$, and a regular ltd d for the inputs to the query. We wish to compute L_X . We proceed in three stages. For each variable Y in $B(\vec{X})$ let us denote by $B_Y(\vec{X})$ the subpattern of $B(\vec{X})$ occurring downstream from Y . We first compute, for each type $a \in \Sigma$ such that $B_Y(\vec{X})$ is satisfiable at a , the tightening $tighten(a, d, B_Y(\vec{X}))$. We next consider the path in $B(\vec{X})$ leading from the root to X . This is of the fol-



In the above, each of the

l_i and r_i may contain additional variables, with their own downstream patterns. The language for L_X is computed by an extension of the technique used for the example query q' . Essentially, the i th step of the algorithm computes the language $L_{X_{i+1}}$ using the language L_{X_i} computed in the previous step. However, this construction of the grammar for L_X is complicated by three main factors:

- the transducer performing horizontal matching must take into account the presence of other variables with their own downstream patterns in l_i and r_i . For each such variable Y , the transducer has to take into account whether the pattern $B_Y(\vec{X})$ is satisfiable or valid at each type against which Y is matched.
- The transducer must perform a tightening step when each variable X_i is matched against some type a . If $B_{X_i}(\vec{X})$ is valid at a , the transducer outputs the root type corresponding to $tighten(a, d, B_{X_i}(\vec{X}))$ (together with the current state information). If $B_{X_i}(\vec{X})$ is satisfiable but not valid at a , the transducer nondeterministically outputs ϵ or the root type corresponding to $tighten(a, d, B_{X_i}(\vec{X}))$ (again, together with the state information).
- to account for vertical navigation, the nonterminals of the grammar must keep track *simultaneously* of the current possible states in all fsa for the paths p_1, \dots, p_k . Whenever an accepting state of f_{p_i} is reached, the transducer step is applied and the start state of $f_{p_{i+1}}$ is added to the set of possible states.

The development in this section leads to the following main result:

Theorem 4.3 *Given a regular ltd d and a selection loto-ql query q , one can effectively construct a tight specialized context-free ltd for $q(T(d))$.*

The complexity of the construction is EXPTIME in the general case and the size of the inferred ltd is polynomial in the input ltd and exponential in the query. It remains open whether the complexity is tight.

Remark 4.4 *In this section we assumed that input lotos are described by regular ltds. Now suppose that the inputs are described instead by specialized context-free ltds. This would happen if defining a loto-ql view on top of another loto-ql view. Also, the concatenation of multiple sources into a single source is described by a specialized regular ltd. Our inference algorithm and Theorem 4.3 generalize easily to such input ltds.*

Conformance The ltd inference algorithm allows to solve an important related problem: checking *conformance* of a selection view definition to a predefined ltd. This is of interest, for example, when data satisfying some ltd must be translated in a form that satisfies another ltd (see also the discussion in [Suc]). We can show:

Corollary 4.5 *It is decidable, given a regular ltd d , a selection loto-ql query q , and another regular ltd d' , whether $q(T(d)) \subseteq T(d')$.*

The proof uses our inference algorithm, the decidability of whether a context-free language is included in a regular language, and the decidability of inclusion of regular tree languages. The complexity is EXPTIME.

Special cases We have seen that describing the view defined by a loto-ql query on inputs satisfying a given regular ltd requires the use of more powerful context-free ltds. There are however special cases of practical interest when specialized regular ltds are sufficient for describing loto-ql views. The special cases restrict either the input regular ltd or the selection loto-ql query defining the view.

The restriction on the input ltds is quite natural. Let us call an ltd *stratified* if the dependency graph among types is acyclic (the dependency graph for an ltd d has an edge from b to a if b occurs in $d(a)$). For example, (LTD2.6) is stratified but (LTD2.9) is not.

The restriction on queries disallows recursion in vertical navigation⁷. That is, regular expressions occurring as labels of edges do not use Kleene closure. The regular expressions used for horizontal navigation may continue to use Kleene closure. Let us call this class of queries *vertically nonrecursive*. By revisiting the inference algorithm in the previous section for the above special cases, we are able to show the following.

Theorem 4.6 *Given a regular ltd d and a selection loto-ql query q such that d is stratified or q is vertically nonrecursive, one can effectively construct a tight specialized regular ltd for $q(T(d))$.*

The complexity of the construction and the size of the resulting ltd remain exponential.

In both cases just considered, specialization is still generally required. However, it easily seen (using Lemma 2.11), that the resulting views always have a *tightest* regular ltd which can provide an approximate description without specialization, and can be effectively constructed. Moreover, in particular cases, tight regular ltds may exist for the view. It turns out that this can be tested, and a tight regular ltd can be effectively constructed if such exists. Contrast this with the general case, where the existence of a tight (or even tightest) regular ltd for a given view is undecidable (Theorem 4.1).

Corollary 4.7 *(i) Given a regular ltd d and a selection loto-ql query q such that d is stratified or q is vertically nonrecursive, $q(T(d))$ has a tightest regular ltd which can be effectively constructed. (ii) Given a regular ltd d and a selection loto-ql query q such that d is stratified or q is vertically nonrecursive, it is decidable in EXPSpace whether $q(T(d))$ has a tight regular ltd, and if so such an ltd can be effectively constructed.*

⁷Some languages, such as MSL and YATL [PAGM96, CDSS98] do not provide recursive vertical navigation in the first place.

Extensions We briefly discuss how our algorithm can be extended for more powerful queries: (i) selection queries with more powerful selection conditions, and (ii) loto-ql queries with the same selection conditions but constructed answers. In regard to (i), it turns out that our algorithm can be extended to selection queries with much more general selection conditions than those of loto-ql. This is indicated by the following result, shown by an extension of our technique. Recall that Monadic Second Order logic (MSO) is first-order logic extended with set variables.

Theorem 4.8 *Let $\varphi(x)$ be an MSO formula over labeled unranked trees, with one free first-order variable x . For each loto t , let $\{n_1, \dots, n_k\} = \{x \mid t \models \varphi(x)\}$, where $n_1 \dots n_k$ is the order of occurrence of the nodes n_i in the pre-order traversal of t . Let $string_\varphi(t) = \lambda(n_1) \dots \lambda(n_k)$. For each regular tree language R , the string language $string_\varphi(R) = \{string_\varphi(t) \mid t \in R\}$ is context-free.*

The above allows extending our inference algorithm to produce a tight specialized context-free ltd for selection queries whose selection condition can be described in MSO. Note that these coincide with the unary queries over unranked trees definable by the Extended Attribute Grammars of [Nev99] and by the Strong Query Automata of [NS99].

Lastly, consider extension (ii). General loto-ql queries (not restricted to selection queries), construct new lotos using a group-by construct defining nested lists. The ltd inference algorithm can be extended to general loto-ql queries. However, the ltd it produces is sound but no longer tight for the view. The reasons for this failure go beyond the algorithm itself: there can be no tight specialized context-free ltd for views defined by general loto-ql queries, or for that matter by any semistructured query language that we are aware of and is able to construct objects. The sound ltd produced by our algorithm can still be used in a variety of ways. For example, it provides a sufficient test of conformance to a predefined ltd. We outline the extension of the inference algorithm in Appendix D.

5 Conclusion

We presented a Data Type Definition inference algorithm that produces tight specialized context-free DTDs for selection views of XML data. We used lotos and ltds as formal abstractions of XML documents and DTDs. The language loto-ql used for view definitions captures the common core of several query languages that have been proposed for XML. As a practically important side effect, the ltds produced by the inference algorithm can be used to test conformance of selection views to predefined ltds.

Acknowledgement The authors wish to thank Pavel Velikhov and Andreas Yanakopoulos for implementing the DTD inference algorithm for XMAS. They also thank Frank Neven and Moshe Vardi for useful discussions relevant to this material.

References

- [Abl97] S. Abiteboul. Querying semistructured data. In *Proc. ICDT Conf.*, 1997.
- [ABS99] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web*. Morgan Kaufman, 1999.
- [AM98] G. Arocena and A. Mendelzon. WebOQL: Restructuring documents, databases, and webs. In *Proc. ICDE Conf.*, 1998.
- [AQM⁺97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The LOREL query language for semistructured data. *International Journal on Digital Libraries*, 1(1), 1997.
- [AV97] S. Abiteboul and V. Vianu. Regular path queries with constraints. In *Proc. PODS Conf.*, 1997.
- [B⁺99] C. Baru et al. XML-based information mediation with mix. In *Demonstrations Program of ACM SIGMOD Conf.*, 1999.
- [BDFS97] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proc. of the International Conference on Database Theory*, 1997.
- [BDHS96] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. ACM SIGMOD*, 1996.
- [Bi] Bluestone-inc. Visual XML. <http://www.bluestone.com/xml/Visual-XML/>.
- [BKMW] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree languages over non-ranked alphabets. Manuscript, 1998. Available at <ftp://ftp11.informatik.tu-muenchen.de/pub/misc/caterpillars/>.
- [BM99] Catriel Beeri and Tova Milo. Schemas for integration and translation of structured and semi-structured data. In *Int'l. Conf. on Database Theory*, 1999.
- [BPSM] T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation. Latest version available at <http://www.w3.org/TR/REC-xml>.
- [Bun97] P. Buneman. Tutorial: Semistructured data. In *Proc. PODS Conf.*, 1997.
- [CD] J. Clark and S. Deach. Extensible stylesheet language (xsl) 1.0, W3C working draft. <http://www.w3.org/TR/WD-xsl>.
- [CDSS98] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion! In *Proc. ACM SIGMOD Conf.*, 1998.
- [CM90] M. Consens and A. Mendelzon. Graphlog: a visual formalism for real life recursion. In *Proc. ACM Symp. on Principles of Database Systems*, 1990.
- [dBV93] J. Van den Bussche and G. Vossen. An extension of path expressions to simplify navigation in object-oriented queries. In *Proc. of Intl. Conf. on Deductive and Object-Oriented Databases (DOOD)*, 1993.
- [DFF⁺] A. Deutch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A query language for XML. Submission to W3C. Latest version available at <http://www.w3.org/TR/NOTE-xml-ql>.
- [FFLS98] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Catching the boat with Strudel: experience with a web-site management system. In *Proc. ACM SIGMOD Conf.*, 1998.
- [FLS98] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proc. PODS Conf.*, 1998.
- [FS98] M. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *Proc. of the International Conference on Data Engineering*, 1998.
- [Gin66] S. Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill Book Co., 1966.
- [GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
- [GS97] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages*, volume 3. Springer Verlag, 1997.
- [GW97] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Proc. VLDB*, 1997.
- [HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [KS95] D. Konopnicki and Oded Shmueli. W3QS: A query system for the World Wide Web. In *Proc. VLDB Conf.*, pages 54–65, Zürich, Switzerland, September 1995.
- [LJM⁺] A. Layman, E. Jung, E. Maler, H. Thompson, J. Paoli, J. Tigue, N. Mikula, and S. De Rose. XML-Data. Available at <http://www.w3.org/TR/1998/NOTE-XML-data>.
- [MD] E. Maler and S. DeRose. XML pointer language (XPointer). <http://www.w3.org/TR/1998/WD-xptr-19980303>.

- [Mit90] J. Mitchell. Type systems for programming languages. *Handbook of Theoretical Computer Science*, 2:367–458, 1990.
- [Mit96] J. Mitchell. *Foundations of Programming Languages*. MIT Press, 1996.
- [MP] K. Munroe and Y. Papakonstantinou. BBQ: A visual interface for integrated browsing and querying of XML. Available at <http://www.db.ucsd.edu/publications/BBQ.pdf>.
- [MS99] T. Milo and D. Suciu. Type inference for queries on semistructured data. In *Proc. PODS Conf.*, pages 215–226, 1999.
- [MSV] T. Milo, D. Suciu, and V. Vianu. Type-checking for XML transformers. Submitted to PODS 00.
- [MW95] A. Mendelzon and P. Wood. Finding regular simple paths in graph databases. *SIAM J. Comp.*, 24(6), 1995.
- [MZ98] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. VLDB Conf.*, 1998.
- [NAM98] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *Proc. ACM SIGMOD Conf.*, 1998.
- [NdB98] F. Neven and J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. In *Proc. PODS Conf.*, pages 11–17, 1998.
- [Nev99] F. Neven. Extensions of attribute grammars for structured document queries. In *Proc. DBPL Conf.*, 1999.
- [NS99] F. Neven and T. Schwentick. Query automata. In *Proc. PODS Conf.*, pages 205–214, 1999.
- [NUWC97] S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Representative objects: Concise representations of semistructured, hierarchical data. In *Proc. ICDE*, 1997.
- [PAGM96] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proc. VLDB Conf.*, 1996.
- [PV] Y. Papakonstantinou and P. Velikhov. The use and computation of specialized dtDs in the MIX mediator system. Available at <http://www.db.ucsd.edu/publications/UseComputeDTD.pdf>.
- [PV99a] Y. Papakonstantinou and V. Vassalos. Query rewriting for semistructured data. In *Proc. ACM SIGMOD Conference*, 1999.
- [PV99b] Y. Papakonstantinou and P. Velikhov. Enhancing semistructured data mediators with document type definitions. In *Proc. ICDE Conf.*, 1999.
- [Si] SoftQuad-inc. XMetal editor. <http://www.sq.com/products/xmetal/>.
- [Suc] D. Suciu. An overview of semistructured data. *SIGACT News* 29:4, pp. 28-38, December 1998.
- [VLP00] P. Velikhov, B. Ludascher, and Y. Papakonstantinou. Navigation-driven evaluation of virtual mediated views. In *Proc. EDBT Conf.*, 2000.

Appendix

A Regular tree languages and tree automata

We informally review the notion of regular tree language and tree automaton. Tree automata are devices whose function is to accept or reject their input, which we assume is a complete binary tree with nodes labeled with symbols from some finite alphabet Σ . There are several equivalent variations of tree automata. A non-deterministic top-down regular tree automaton over Σ has a finite set Q of states, including a distinguished initial state q_0 and an accepting state q_f . In a computation, the automaton labels the nodes of the tree with states, according to a set of rules, called *transitions*. An internal node transition is of the form $(a, q) \rightarrow (q', q'')$, for $a \in \Sigma$. It says that, if an internal node has symbol a and is labeled by state q , then its left and right children may be labeled by q' and q'' , respectively. A leaf transition is of the form $(a, q) \rightarrow q_f$ for $a \in \Sigma$. It allows changing the label of a leaf with symbol a from q to the accepting state q_f . Each computation starts by labeling the root with the start state q_0 , and proceeds by labeling the nodes of the trees non-deterministically according to the transitions. The input tree is accepted if *some* computation results in labeling all leaves by q_f . A set of complete binary trees is *regular* iff it is accepted by some top-down tree automaton. Regular tree languages have similar properties to regular string languages, including closure properties and decidability of emptiness (in PTIME), inclusion (in EXPTIME), etc. Regular languages of finite binary trees are surveyed in [GS97]. An analogous extension to the case of unranked trees is discussed in [BKMW]. The above results extend to the unranked case.

B Construction of a tightened specialized ltd

We outline the construction of a tight specialized ltd for the set of lotos rooted at a , satisfying a given ltd d , for which there exists a downward path in p from the root. Let f_p be an fsa over Σ accepting p , with start state s , set of final states F , and state transition function δ . For each state h in f_p , let p_h be the regular language accepted by f_p with start state h . The specialized ltd is $\mathbf{d}' = (\Sigma, \Sigma', d', \mu)$ where:

- $\Sigma' = \Sigma \cup \{\langle h, b \rangle \mid h \in \text{states}(f_p), b \in \Sigma, \text{ and } p_h \text{ is satisfiable at } b\}$;
- $\mu(\langle h, b \rangle) = b$ and $\mu(b) = b$ for $b \in \Sigma$ and $h \in \text{states}(f_p)$;


- $d'(\text{root}) = \langle s, a \rangle$;
- $d'(b) = d(b)$ for $b \in \Sigma$;
- $d'(\langle f, b \rangle) = d(b)$ if f is accepting;
- $d'(\langle h, b \rangle) = \mu^{-1}(d(b)) \cap \Sigma^* \Sigma'_{\text{next}} \Sigma^*$ if h is not accepting, where $\Sigma'_{\text{next}} = \{\langle h', a \rangle \mid h' = \delta(h, a) \text{ and } p_{h'} \text{ is satisfiable at } a\}$.

Note that the last item simply ensures that the content of $\langle h, b \rangle$ has at least one type allowing to continue successfully along a path in p . The following is easily seen:

Lemma B.1 *A loto with root of type a satisfies the specialized ltd \mathbf{d}' iff it satisfies d and there exists a path in p starting at its root.*

Thus, p is valid at the root of lotos satisfying \mathbf{d}' . We denote the specialized ltd \mathbf{d}' tightening a as described by $\text{tighten}(a, d, p)$.

C Examples for inference algorithm

Example C.1 *Consider the query and the source described in Example 3.1. The automaton f_p of the path $p = \text{section}^*$ is  section. The automaton also contains a sink state \perp (omitted) where all non-indicated transitions are directed. The procedure described in Section 4 yields the following grammar for L_X :*

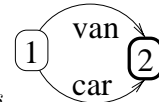
$$\begin{aligned} \langle 1, \text{root} \rangle &\rightarrow \text{intro}.\sigma_1(\text{intro}).(\sigma_1(\text{section}))^*.\text{conc}.\sigma_1(\text{conc}) \\ \langle 1, \text{section} \rangle &\rightarrow \text{intro}.\sigma_1(\text{intro}).(\sigma_1(\text{section}))^*.\text{conc}.\sigma_1(\text{conc}) \\ \langle 1, \text{intro} \rangle &\rightarrow \epsilon \\ \langle 1, \text{conc} \rangle &\rightarrow \epsilon \end{aligned}$$

Since $\sigma_1(\text{section}) = \{\langle 1, \text{section} \rangle\}$ and $\sigma_1(\text{intro}) = \sigma_1(\text{conc}) = \{\epsilon\}$, the above simplifies to

$$\begin{aligned} \langle 1, \text{root} \rangle &\rightarrow \text{intro}(\langle 1, \text{section} \rangle)^* \text{conc} \\ \langle 1, \text{section} \rangle &\rightarrow \text{intro}(\langle 1, \text{section} \rangle)^* \text{conc} \end{aligned}$$

This generates essentially the words of well-balanced parenthesis (where intro serves as open parenthesis and conc as closed parenthesis).

Example C.2 *Consider the query and source ltd (call it d) of Example 3.2. The automaton f_p for*



$p = (\text{van} + \text{car})$ is *The language p_1 accepted by f_p starting from state 1 is $\text{van} + \text{car}$, the language p_2 is ϵ , and $p_\perp = \emptyset$. The grammar for L_D*

has start symbol $\langle 1, \text{root} \rangle$ and the following productions:

$$\begin{aligned} \langle 1, \text{root} \rangle &\rightarrow \sigma_{\perp}(\text{truck}) + \sigma_2(\text{van}) + \\ &\quad \sigma_{\perp}(\text{RV}) + \sigma_2(\text{car}) \\ \langle 2, \text{van} \rangle &\rightarrow (\text{dealer}.\sigma_{\perp}(\text{dealer}))^* \\ \langle 2, \text{car} \rangle &\rightarrow (\text{dealer}.\sigma_{\perp}(\text{dealer}))^* \end{aligned}$$

where $\sigma_2(\text{van}) = \{\langle 2, \text{van} \rangle\}$, $\sigma_2(\text{car}) = \{\langle 2, \text{car} \rangle\}$, $\sigma_{\perp}(\text{truck}) = \sigma_{\perp}(\text{RV}) = \{\epsilon\}$ (since p_2 is valid at van and car and p_{\perp} is unsatisfiable at truck and RV). Thus, the language L_D is simply dealer^* . Finally, we must modify L_D to take into account the conditions that the downwards pattern imposes on dealer trees. The specialized ltd $\text{tighten}(\text{dealer}, d, \text{ad}.\text{UsedAd})$ has root dealer^u with content $\text{ad}^* \text{ad}^u \text{ad}^*$ where ad^u has content UsedAd . Thus, the final substitution σ is defined by $\sigma(\text{dealer}) = \text{dealer}^u$. In summary, the final specialized ltd for the query is:

$$\begin{aligned} \text{root} &: (\text{dealer}^u)^* \\ \text{dealer}^u &: \text{ad}^* \text{ad}^u \text{ad}^* \\ \text{ad}^u &: \text{UsedAd} \\ \text{ad} &: \text{UsedAd} + \text{NewAd} \end{aligned}$$

D Loto-ql with constructed answers

We outline an extension of our ltd inference algorithm for general loto-ql queries (not restricted to selection queries), which construct new lotos using a powerful group-by construct defining nested lists. The inference algorithm produces a *sound* context-free specialized ltd for the answers to such queries.

Loto-ql with constructed answers A general loto-ql query is of the form

$$\begin{aligned} &\text{construct } H(\vec{X}) \\ &\text{where } B(\vec{X}) \end{aligned}$$

In the above, $B(\vec{X})$ is called the *body* of the query, and $H(\vec{X})$ the *head*. The body is as described for selection loto-ql queries. The head is an ordered labeled tree. It specifies how to build a new loto using the bindings provided by the body of the query. The head is itself a loto, augmented with so called *group-by* labels. Ignoring the group-by labels, which we discuss shortly, the nodes of the loto are labeled by a symbol in the alphabet, or by a *term*. A term is a variable X or an expression $\text{type}(X)$ (denoting the type of $\text{root}(X)$). The set of terms using variables from the body $B(\vec{X})$ is denoted $\text{Terms}(B(\vec{X}))$. The root is always labeled by a symbol. Internal nodes can only be labeled by a symbol or by a term $\text{type}(X)$. Thus, only leaves of the loto can be labeled by X (recall that variables X bind to entire subtrees in the input). We usually denote terms by T_1, T_2, \dots, T_k . We call a tree as above

parameterized (by the terms it contains), and make the parameters explicit by writing $t(T_1, \dots, T_k)$.

We next describe group-by labels. Each group-by label is a (possibly empty) sequence of distinct terms in $\text{Terms}(B(\vec{X}))$. Group-by labels are denoted $[T_1 \dots T_k]$. Similarly to logical quantification, the *scope* of a group-by label of a node is the subtree rooted at that node. A group-by labeling must satisfy the following: (i) the root has group-by label ϵ ; ⁸, and (ii) every occurrence of a term T in the head is in the scope of some group-by label containing T . We now have all the ingredients for defining the head of a query: the head consists of a parameterized tree together with a group-by labeling.

Given a query with body $B(\vec{X})$ and head $H(\vec{X})$, the answer to the query on given input is constructed from the set of bindings \mathcal{B} of variables satisfying the pattern $B(\vec{X})$ (see definition of binding in Section 3). Each binding $\beta \in \mathcal{B}$ extends to terms $\text{type}(X)$ in the obvious way: if t is an input loto with labeling λ and β is a binding for the variables, then $\beta(\text{type}(X)) = \lambda(\text{root}(\beta(X)))$. The answer to the query is a loto constructed by structural recursion on $H(\vec{X})$ as follows. The recursion uses partial bindings of the variables. The partial binding associated with the root is empty. Each subtree $t(T_1 \dots T_k)$ whose root has group-by label $[T_1 \dots T_k]$ and whose ancestors group-by variables are instantiated by a partial binding α is recursively replaced by a list of subtrees consisting of one isomorphic copy of $t(\beta(T_1) \dots \beta(T_k))$ for each restriction β of some binding in \mathcal{B} that extends α to T_1, \dots, T_k . The order of the subtrees in the list is given by the lexicographic order of the bindings (for terms of the form $\text{type}(X)$, assume a default ordering of the types). In view of the definition of group-by labeling, it is clear that the above procedure yields a loto.

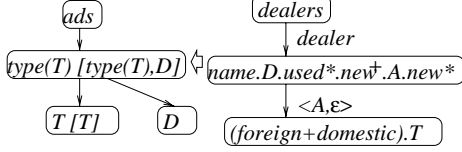
Example D.1 Consider a “dealers” input containing car ads, partially described by the following ltd piece:

$$\begin{aligned} \text{root} &: \text{dealers}; \\ \text{dealers} &: \text{dealer}^*; \quad \text{dealer} : \text{name}, \text{used}^*, \text{new}^*; \\ \text{used} &: (\text{foreign} + \text{domestic} + \text{sedans} + \text{RVs}); \\ \text{new} &: (\text{foreign} + \text{domestic} + \text{sedans} + \text{RVs}); \\ \text{foreign} &: \text{model}, (\text{year} + \epsilon); \\ \text{domestic} &: \text{model}, (\text{year} + \epsilon); \end{aligned}$$

Now consider the query below (the head is left of the arrow). The query retrieves the domestic and foreign new car ads, which bind to T , along with the names D of the corresponding dealers. The answer restructures the input by classifying the ads into a list of “domestic” lotos, which is followed by a list of “foreign” lotos (since $\text{type}(T)$ can only be “domestic” or “foreign”). In particular, there is one “domestic” loto for each dealer who sells at least one domestic car and simi-

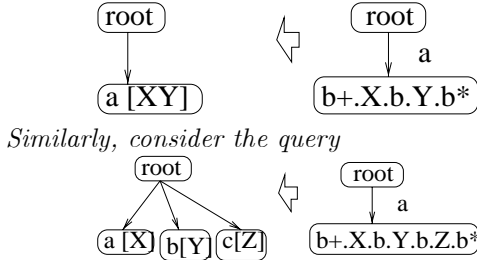
⁸Empty labels are omitted in examples.

larly for “foreign”. Each “domestic” loto contains a list of all “domestic” ads published by the specific dealer. The list is followed by the name D of the dealer; the “foreign” loto is similar. Note that we pick the full ads but only the dealer name nodes in the answer.



Ltd inference for general loto-ql queries The ltd inference algorithm we described for selection queries can be extended to general loto-ql queries. However, the ltd it produces is sound but no longer tight for the view. The sound ltd it produces can still be used in a variety of ways. For example, it provides a sufficient test of conformance to a predefined ltd. Before outlining the extension of the inference algorithm, we briefly discuss its failure to provide a tight ltd for general loto-ql queries. Unfortunately, the reasons for this failure go beyond the algorithm itself: there can be no tight specialized context-free ltd for views defined by general loto-ql queries. This is illustrated next.

Example D.2 *The following query always produces in the answer lists of length $n(n-1)/2$ which cannot be described by a context-free language.*



It generates lists of the form $a^n b^n c^n$ which is not a context free language.

We next describe the extension of our ltd inference algorithm to general loto-ql queries. We use the notation developed in our presentation of the algorithm in Section 4. Let q be a loto-ql query and d a regular ltd for the input. When considering the group-by structure of the query head, it will be necessary to compute the languages L_X for variables X in the context of a partial instantiation of the terms in $Terms(B)$ (the ones in whose group-by scope X occurs). The inference algorithm for selection queries can be adapted to this case by first tightening the input ltd d with respect to the partially instantiated body. For a partial assignment λ of types to variables, let $L_X(\lambda)$ denote the language L_X in the context of λ . More precisely, $L_X(\lambda)$ is L_X for the input ltd tightened with respect to $B(\lambda(\vec{X}))$, in which each X in the domain of λ is

replaced by $\lambda(X)$ in $B(\vec{X})$. Similarly, $\Sigma_X(\lambda)$ denotes the possible types of roots of subtrees to which X can bind in the context of λ .

To define a sound specialized ltd for the answers to q , it is clearly sufficient to infer the language L_n corresponding to each node n in the head. Recall that each node n generates a list, in accordance to its group-by label. The list also depends on the context provided by each assignment λ of types to the variables \vec{Y} in whose group-by scope n occurs. Let λ be a fixed type assignment for \vec{Y} . First, suppose n has empty group-by label. If n is a symbol a , $L_n = \{a\}$. If n is a term X or $type(X)$, $L_n = \Sigma_X(\lambda)$.

Now consider the more interesting case when n has nonempty group-by label $\vec{Z} = [Z_1 \dots Z_k]$. We need to consider the possible types of the bindings for \vec{Z} in the lexicographic order of the bindings. This can be viewed as a language $L_{\vec{Z}}$ over an alphabet with symbols of the form $[Z_1 : a_1 \dots Z_k : a_k]$, where the a_i are types. If $k = 1$, the language $L_{\vec{Z}}$ is $L_{Z_1}(\lambda)$. If $k > 1$, computing the language $L_{\vec{Z}}$ is more complicated, since it is not determined by the languages for each individual Z_i . This is illustrated in Example D.2 (ii), where $L_X = L_Y = L_Z = b^+$ but L_{XYZ} is constrained to contain, for each of n occurrences of X , n occurrences of Y and n occurrences of Z . However, one can use the languages for each Z_i to obtain an approximation $\bar{L}_{\vec{Z}}$ containing $L_{\vec{Z}}$. The language $\bar{L}_{\vec{Z}}$ is computed from the languages $L_{Z_i}(\lambda_i)$, $1 \leq i \leq k$, where each $L_{Z_i}(\lambda_i)$ is defined relative to a context λ_i (augmenting λ) provided by a type assignment for Z_j , $j < i$. The language $\bar{L}_{\vec{Z}}$ is obtained using an appropriate sequence of substitutions. For instance, if $k = 2$, the language $\bar{L}_{Z_1 Z_2}$ is $\tau(L_{Z_1})$ where τ is the substitution on Σ_{Z_1} defined by

$$\tau(b) = \{[Z_1 : b, Z_2 : b_1] \dots [Z_1 : b, Z_2 : b_m] \mid b_1 \dots b_m \in L_{Z_2}(\lambda_b)\}$$

where λ_b assigns b to Z_1 .

To make the context for $\bar{L}_{\vec{Z}}$ explicit, we denote the language $\bar{L}_{\vec{Z}}$ in context λ by $\bar{L}_{\vec{Z}}(\lambda)$.

Now suppose n is a symbol a . Then $L_n(\lambda)$ is contained in $h(\bar{L}_{\vec{Z}}(\lambda))$ where h is the homomorphism mapping every symbol of $\bar{L}_{\vec{Z}}(\lambda)$ to a . Next, suppose n is a variable Z_i in \vec{Z} . Then $L_n(\lambda)$ is contained in $h(\bar{L}_{\vec{Z}}(\lambda))$ where h is the homomorphism defined by $h([Z_1 : a_1 \dots Z_k : a_k]) = a_i$ for each symbol $[Z_1 : a_1 \dots Z_k : a_k]$ of $\bar{L}_{\vec{Z}}(\lambda)$. The case when n is a term $type(Z_i)$ is similar. The inference mechanism we described proceeds by structural recursion, with the appropriate context λ passed from parents to children. As a final step, the languages with respect to the various contexts are used to construct a single specialized ltd encompassing a “case analysis” by the relevant contexts. It can be shown that the ltd constructed above is sound for $q(T(d))$. For instance, our

algorithm yields in Example D.2 (ii) the ltd describing the content of the root as $a^*b^*c^*$.

In addition to soundness, the ltd produced by our algorithm satisfies a practically appealing notion of tightness. Suppose each group-by label in q uses only a single term. Then the algorithm allows to infer tight ltds for the lists induced by each node in the head of q . We call such an ltd *locally tight*. Local tightness is practically significant, because it provides precise descriptions of portions of the answer which are intuitively meaningful.