DTD Inference for Views of XML Data (AMERICA251)

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

Abstract

We study the inference of DTDs for views of XML data, using an abstraction that focuses on document content structure. The view definition language 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 of 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), and we consider more powerful views that construct complex documents. Finally we consider related problems such as checking conformance of a view definition with a predefined DTD.

1 Introduction

Current database systems are faced with a qualitative jump in the complexity of the data they have to manage. New application domains require handling heterogeneous data, distributed among multiple data sources, and whose structure is not regular or completely known. This is a significant departure from traditional databases handling highly-structured data described uniformly by a rigid schema. Recent research on semistructured data has attempted to address this challenge (see the surveys [Bun97, Abi97, Suc]). The emergence of 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 Definitions[1] (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 integrated views and querying XML data. QBE-style query interfaces [BGL+] may use DTDs to display the "schema" of an integrated view and allow users to navigate it. DTDs may help in the design of the storage structures. Mediators, which create integrated

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

views by selecting and restructuring source data, may use DTDs to optimize the queries that they send to the sources [BGL+]. Finally DTDs may guide the production of style sheets, such as XSL scripts [CD], that display XML documents as browser-compatible HTML documents.

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 integration software development. The goal of this paper is to present algorithms and results for 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. Our abstraction leaves out many bells and whistles of XML. The extended version of the paper [PV] compares in detail lotos with XML 1.0 [BPSM] and discusses how our results can be extended to the full XML specification.

To study DTD inference for views we introduce a view definition language that queries labeled ordered 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. To the best of our knowledge, horizontal regular expressions are a novel semistructured query language feature which seamlessly couples with the well-known regular path expressions for vertical navigation [Suc, Abi97, Bun97, CM90, MW, dBV93, AQM+97, BDHS96, FFLS98, FLS98, AM98, CDSS98, DFF+, KS95, AV97]. In a sense, our language enhances the horizontal navigation functionality of the XPointer standard (available at `w3c.org`) and incorporates it into a semistructured query language. The variable bindings extracted by the tree pattern are used to construct the answer. We focus for most of the paper on *selection queries*, which extract from the input the subtrees to which one of the variables in the tree pattern binds. We then consider queries that construct more complex documents using several variables. This is done using a powerful group-by feature, which generalizes SQL's group-by and allows the construction of nested lists. 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). The formal language provides the theoretical basis of XMAS, a language for defining XML views implemented and used in the MIX project [BGL+]. XMAS additionally considers XML details that are left out in the labeled ordered tree abstraction.

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. Clearly, tight ltds are most desirable, as they provide the maximum information on the structure of documents in the view. In particular, a tight ltd describes only those document structures that can occur 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].

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.

Our ltd inference algorithms for selection queries provides useful insight into the technical problems involved in producing ltds that are as precise as possible. We next consider how the techniques we have developed can be adapted to extensions of the model and view definition language. First, we look at views that construct complex documents using a "group-by" construct. The ltd inference algorithm for selection views can be extended to provide a sound (but no longer tight) description of such views. Intuitively, tightness is lost due to dependencies among variables that cannot be captured by specialized context-free ltds. Nevertheless, for specific classes of group-by lists the derived ltd is locally tight, in the sense that individual lists of the tree are precisely described. (However, dependencies across lists are not captured.)

Lastly, we consider additional features such as querying XML attribute values and text. In this case, there can be no tight ltd (or DTD) for the view because ltds (and DTDs) do not in general place constraints on string content. However the inferred DTD for a selection view is tight in a weaker sense: every type structure allowed by the ltd is realized by some document in the view.

Related work Although the traditional assumption in dealing with semistructured data is that there is no explicit schema, there has been considerable effort in recovering schematic information from data. Notable approaches include the representative objects and dataguides [NUWC97, GW97] and the graph schemas of [BDFS97]. The problems they address are orthogonal to ours. Inference of schemas for views is not considered. However, it is interesting to compare the specification power of ltds with that of dataguides and graph schemas, and we do so next.

Dataguides
[NUWC97, GW97] Given a semistructured OEM database [QRS+95], dataguides are structures that can efficiently answer the query: *what types can be reached from an object of type o following a path p?* Interestingly, the dataguides themselves are represented as OEM objects.[2]

Graph Schemas
[BDFS97] provide a "schema" based on graphs where the edges are labeled with predicates. A database satisfies the graph schema if there is a homomorphism of the database (itself a labeled graph) to the graph schema, such that the edge labels of the database satisfy the edge predicates they are mapped to. [BDFS97] also provide computational properties of graph schemas.

In order to meaningfully compare the specification power of ltds with that of data guides and graph schemas, we have to first level the field by ignoring the obvious differences that immediately render the different approaches incomparable (e.g., ltds work on an ordered graph model, and graph schemas use arbitrary edge predicates). Then it is quite easily seen that specialized ltds are strictly more powerful than both dataguides and graph schemas. One important difference is that ltds can require the *presence* of certain patterns in the data, which cannot be done with dataguides or graph schemas.

Note that both graph schemas and dataguides implicitly allow a form of specialization. A type a may be assigned different subobjects depending on its ancestors. Consequently, they can describe sets of databases that cannot be described by plain ltds; nevertheless, plain ltds can also describe sets of databases that graph schemas or dataguides cannot.

The use of dataguides for schema discovery and query optimization is discussed in [NUWC97, GW97], and the use of graph schemas for query optimization is shown in [FS98]. DTDs are used in [MZ98] for schema matching, which, in turn, is used for data conversion. Specialized ltds can also support these activities, as proposed in [BGL+]. 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.

Note that the "patterns" in [CDSS98] are also a form of dataguide. A limited form of inference can be accomplished by inferring the patterns that view variables may bind to. 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.

Schema inference in views is related in spirit to inferring dependencies in relational views. This was first investigated in [Klu80, KP82] for relational algebra views, and later in [AH88] for Datalog views.

An informal discussion of problems raised by schema inference in views of semistructured data is presented in [PV99]. 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 views defined by

[2] However applying the view on the dataguide does *not* produce the dataguide of the view. Indeed, it is impossible to derive the dataguide of the view from the dataguide of the database.

a simple class of queries using only one variable, no horizontal navigation, and no recursive path expressions.

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. Lastly, Section 5 considers ltd inference in the context of extensions to the model the view definition language.

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 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 transducers.

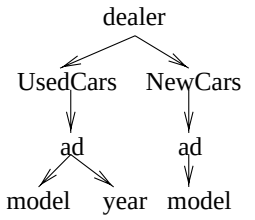
Labeled ordered tree objects A labeled ordered tree object (loto) is our abstraction of an XML document. Each node represents an XML element and is labeled by the element's name (type). The list of children of a node represents the sequence of elements that make up the content of the node, labeled by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

Definition 2.1 *A* labeled ordered tree object (loto) *over alphabet*[3] 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 ordered.*

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).*

```
<dealer>
 <UsedCars>
  <ad>
   <model>Honda</model>
   <year>92</year>
  </ad>
 </UsedCars>
 <NewCars>
  <ad>
   <model>BMW</model>
  </ad>
 </NewCars>
</dealer
```



(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*

[3] We only consider finite alphabets.

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 $d(root)$.

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 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, denoted $t \models d$, if the root has type $d(root)$ 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 $\emptyset$ – e.g., for* model *and* year *below.*

(LTD2.6)

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

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 be $\emptyset$ or must contain ϵ. In the example, $d(model) = d(year) = \emptyset$, and $d(UsedCars), d(NewCars)$ contain ϵ.

There is an interesting connection between regular ltds and context free grammars. Clearly, a regular ltd is somewhat similar to a set of productions in a CFG, each of which specifies how an element can be expanded. In an ltd the right-hand side of a production is a regular expression rather than a single word as in grammars, but the difference is not crucial.

```
<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

In some sense, the set of lotos satisfying a given ltd is a CFL. To make this more precise, we define a standard in-line representation of lotos by structural recursion as follows. If t is a loto whose root n has the sequence of subtrees $tree(n_1) \dots tree(n_k)$, then $\textit{in-line}(t) = \lambda(n)[\textit{in-line}(tree(n_1)) \dots \textit{in-line}(tree(n_k))]$.[4] If t consists of a single leaf node n then $\textit{in-line}(t) = \lambda(n)$. It easy to see that for every regular ltd d, the language $\textit{in-line}(T(d)) = \{\textit{in-line}(t) \mid t \in T(d)\}$ is context-free.

Although $\textit{in-line}(T(d))$ is a CFL, there is an important distinction between lotos and arbitrary CFLs. The lotos satisfying an ltd are analogous to the *derivation trees* of CFG rather than the words generated by grammars. This has important consequences. Indeed, although equivalence of CFGs is undecidable, it is easily decidable if two grammars generate the same derivation trees. Similarly, questions involving relationships between regular ltds are generally de-

[4] It is easy to see that *in-line* produces the string representation of an XML document – modulo the difference in the syntax of delimiters.

cidable.

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:

(i) For each $a \in \Sigma$, let L_a be the language consisting of all words $\lambda(n_1) \ldots \lambda(n_k)$ for which $n_1 \ldots n_k$ 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.*

$$\left[\begin{array}{l} root : \text{UsedCars}; \\ \text{UsedCars} : \text{ad}*; \\ \text{ad} : \text{model}, \text{year}; \end{array}\right] \qquad \left[\begin{array}{l} root : \text{NewCars}; \\ \text{NewCars} : \text{ad}*; \\ \text{ad} : \text{model}; \end{array}\right]$$

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^* *A tight*
ad : (model, year) + model;

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 **d,e,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; ad^n : model;

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

root : all;
all : (UsedCars, NewCars);
UsedCars : $(\text{ad}^u)^*$; NewCars : $(\text{ad}^n)^*$;
ad^u : model, year;
ad^n : model;

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

How hard is it to check if a loto satisfies a specialized ltd? Let $\mathbf{d} = \langle \Sigma, \Sigma', d, \mu \rangle$ be a specialized

ltd and t a loto over Σ. Clearly, t satisfies **d** iff

$$\text{in-line}(t) \in \ \mu(\textit{in-line}(T(d)))$$

where μ is extended to $\Sigma' \cup \{[,]\}$ by the identity on $\{[,]\}$. Since, as noted earlier, $\textit{in-line}(T(d))$ and therefore $\mu(\textit{in-line}(T(d)))$ are context-free, this reduces to checking membership of a word w in a CFL, which is $O(|w|^3)$ for a fixed grammar, using the Cocke-Younger-Kasami algorithm (there are more efficient parsing algorithms, see [HU79]). Note that a CFG for $\mu(\textit{in-line}(T(d)))$ is computable from d in linear time, but then has to be brought to Chomsky normal form before the CYK algorithm can be applied.

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. Loto-ql is the formal basis of XMAS, a language which has been implemented for the MIX mediation project. Like [CDSS98, AM98], our language handles order explicitly. We will study the ltd inference problem in loto-ql views. We start with views defined by *selection* loto-ql queries, which we define in this section. Full loto-ql queries will be defined in Section 5.

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[5] 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 $p_0.X_1.p_1.X_2.p_2 \ldots X_k.p_k$, $k \geq 0$, 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.

More 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 $\beta(root(B)) = root(t)$ and for each edge in B labeled $\langle X, p\rangle$ with target node labeled $p_0.X_1.p_1.X_2.p_2 \ldots X_k.p_k$ there exists a path with nodes $x_0 \ldots x_n$ in t where:

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

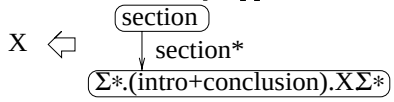
[5] Regular path expressions for vertical navigation are a common feature of almost all semistructured query languages [CM90, MW, FLS98, dBV93, AQM+97, BDHS96, FFLS98, AM98, CDSS98, DFF+, KS95, AV97].

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 $p_0.X_1.p_1.X_2.p_2 \ldots X_k.p_k$ of the target node of the edge provides a pattern matched against the sequence of children of x_n.

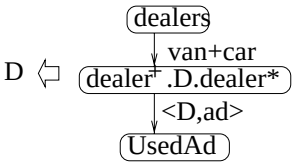
Note that the bindings for a given variable are naturally ordered by a depth-first left-to-right traversal of the input loto.

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.*

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 provides natural yet powerful primitives, and 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:

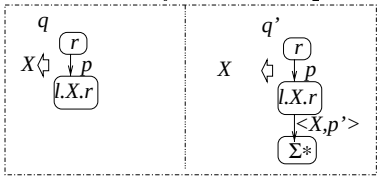
(i) adding a *specialization* mechanism, and

(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 input lotos satisfy a given regular ltd[6]. 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 describe 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

[6] The 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).

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* PSPACE *whether p is valid at a.*

Satisfiability of p at a can be checked in PTIME by testing non-emptiness of the language accepted by an nfsa constructed from p, d and a. Validity is less straightforward and can be tested in PSPACE using an alternating polynomial-time Turing machine. To see how the PTIME alternation arises, let $a \in \Sigma$, $w \in d(a)$, and Σ_w be the set of symbols occuring in w. Note that p is valid at a iff for every Σ_w, $w \in d(a)$, there exists a type $a' \in \Sigma_w$ such that p is valid at a'. The recursive application of this test gives rise to the alternating PTIME algorithm. Using the equivalence of alternating PTIME and PSPACE [CKS81], this yields a PSPACE algorithm.

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. We outline the construction of the tightened specialized ltd. 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:

- $\Sigma' = \Sigma \cup \{\langle h, b\rangle \mid h \in states(f_p), b \in \Sigma,$ and p_h is satisfiable at $b\}$;
- $\mu(\langle h, b\rangle) = b$ and $\mu(b) = b$ for $b \in \Sigma$ and $h \in states(f_p)$;
- $d'(root) = \langle s, a\rangle$;
- $d'(b) = d(b)$ for $b \in \Sigma$;
- $d'(\langle f, b\rangle) = d(b)$ if f is accepting;
- $d'(\langle h, b\rangle) = \mu^{-1}(d(b)) \cap \Sigma^* \Sigma'_{next} \Sigma^*$ if h is not accepting, where $\Sigma'_{next} = \{\langle h', a\rangle \mid h' = \delta(h, a)$ and $p'_{h'}$ 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 4.3 *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 $tighten(a, d, p)$.

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 below.

To define L_X, let us first focus on the nodes which are parents of $root(X)$, i.e. they are reachable by a path in p. Let us denote the corresponding language by L_p. The language L_p is defined by the following CFG G (for convenience, we use productions whose right-hand sides are regular expressions over terminals and nonterminals, with the obvious meaning). 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. The productions of G are:

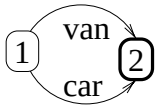
- $\langle h, a\rangle \rightarrow \sigma_h(d(a))$ where h is a state in f_p, $a \in \Sigma$, and σ_h is a substitution defined as follows. For $b \in \Sigma$ and $h' = \delta(h, b)$:
  - $\sigma_h(b) = \{\langle h', b\rangle\}$ if $p_{h'}$ is valid at b,
  - $\sigma_h(b) = \{\epsilon, \langle h', b\rangle\}$ if $p_{h'}$ is satisfiable but not valid at b, and
  - $\sigma_h(b) = \{\epsilon\}$ if $p_{h'}$ is unsatisfiable at b.
- $\langle f, a\rangle \rightarrow a$ for each accepting state f of f_p.

It is easily verified that $L(G) = L_p$. To obtain L_X from L_p, we need one more step, where the pattern $l.X.r$ is matched against the content of nodes reachable by p. For each $a \in \Sigma$, the words resulting by matching $l.X.r$ against the content of a form a language r_a which can be computed from $d(a)$ using a nondeterministic finite-state transducer. The transducer works as follows. Given an input word w, it outputs 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 transducers [Gin66], r_a is regular. Let us call $hmatch(l.X.r, d(a))$ the procedure outputting r_a by matching $l.X.r$ against $d(a)$ using the transducer described above.

As an additional subtlety, a transducer step similar to the above has to also be applied to righthand sides of productions of G with lefthand sides of the form $\langle h, a\rangle$ where h is an accepting state of f_p (details are omitted).

Lastly, $L_X = \rho(L_p)$ where ρ is the substitution $\rho(a) = r_a$ for each $a \in \Sigma$. Since CFLs are closed under substitution [HU79], L_X is context-free. Indeed, a CFG for it can be effectively constructed from d and q in EXPTIME[7] (polynomial in d and exponential in q). In summary, the view defined by q on inputs satisfying d has a tight context-free ltd constructible in EXPTIME.

Example 4.4 *Let us first consider the query and source ltd (call it d) of Example 3.2. The automaton f_p for $p =$ (van + car) is*



. The automaton also contains a sink state $\perp$ (omitted) where all non-indicated transitions are directed. The language p_1 accepted by f_p starting from state 1 is van+car *and the language p_2 is ϵ.*

[7] The construction can be done in PSPACE by using an nfsa for p rather than the fsa we used for simplicity of presentation.

It is always the case that $p_\perp = \emptyset$. *The grammar* L_p *has start symbol* $\langle 1, \text{dealers}\rangle$ *and the following productions:*

$$\langle 1, \text{dealers}\rangle \to \langle 2, \text{van}\rangle, \langle 2, \text{car}\rangle$$
$$\langle 2, \text{van}\rangle \to \text{van}$$
$$\langle 2, \text{car}\rangle \to \text{car}$$

The rhs of the first production was derived from d(dealers) *by applying the substitution* $\sigma_1(\text{truck}) = \{\epsilon\}$ *and* $\sigma_1(\text{RV}) = \{\epsilon\}$ *(because* $p_\perp = \emptyset$ *so* $p_\perp$ *is unsatisfiable at* truck *and* RV*),* $\sigma_1(\text{van}) = \{\langle 2, \text{van}\rangle\}$, *and* $\sigma_1(\text{car}) = \{\langle 2, \text{car}\rangle\}$ *(because* p_2 *is valid at* van *and* car*).*

The language L_D *is obtained by the substitution* $\rho(L_p)$ *where* $\rho(\text{van}) = \text{hmatch}(\text{dealer}^+.D.\text{dealer}^*, \text{dealer}^*) = \text{dealer}^+$ *and, similarly,* $\rho(\text{car}) = \text{dealer}^+$. *The specialized ltd* $tighten(dealer, d, ad.UsedAd)$ *has root* $dealer^u$ *with content* $ad^*ad^uad^*$ *where* ad^u *has content* $UsedAd$. *Thus, the final substitution* σ *is defined by* $\sigma(dealer) = dealer^u$. *In summary, the final specialized ltd for the query is:*

$$root : (dealer^u)^+$$
$$dealer^u : ad^*ad^uad^*$$
$$ad^u : UsedAdad : UsedAd + NewAd$$

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 folllows. $\mathbf{d}'$ 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 $q'(T(d))$.

Putting it all together

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

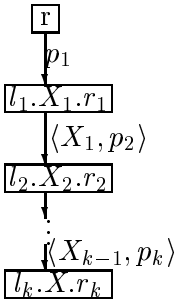
Satisfiability and Tightening Revisited We have seen in the previous discussion that one can test if a regular path expression p is satisfiable or valid at $a \in \Sigma$, given a regular ltd d (see Lemma 4.2). We have also seen how a specialized regular ltd can be constructed to "tighten" a type definition 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, with some increase in the complexity of checking satisfiability. Thus, for each type a, regular ltd d, and loto-ql query body $B(\vec{X})$, all over Σ, one can test in NP whether $B(\vec{X})$ is satisfiable at a, and in PSPACE 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

$L_{X_0} \leftarrow \{r\}$ if the condition is valid
$L_{X_0} \leftarrow r + \epsilon$ if the condition is satisfiable
$L_{X_0} \leftarrow \epsilon$ if the condition is unsatisfiable
for $i = 1, \ldots, k$ %compute L_{X_i}
  for each type a in $L_{X_{i-1}}$
    compute the language L_{p_i} of nodes
      reachable from a following p_i
    for each type b in L_{p_i}
      define the substitution
        $\rho(b) = hmatch^+(l_i.X_i.r_i, d(b)$
    $L^a_{X_i} \leftarrow \rho(L_{p_i})$
  define the substitution $\tau(a) = L^a_{X_i}$
  $L_{X_i} \leftarrow \tau(L_{X_{i-1}})$

Figure 2: The ltd inference algorithm

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 $B(\vec{X})$ occuring 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 following form.



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'. We begin by computing L_{X_1}. As in the case of q', we first compute the language L_{p_1} corresponding to the nodes reachable from r by a path in p_1. Next, we use the routine $hmatch^+$ (extension of $hmatch$) to produce the language L_{X_1} by matching $l_1.X_1.r_1$ against the children of the nodes in L_{p_1}. This is complicated by the presence of other variables with their own downstream patterns in l_1 and r_1. $hmatch^+$ also included the tightening step described below. 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. When X_1 is matched against some type a so that $B_{X_1}(\vec{X})$ is valid at a, the transducer outputs the root type corresponding to $tighten(a, d, B_{X_1}(\vec{X}))$. When X_1 is matched against a so that $B_{X_1}(\vec{X})$ is satisfiable but not valid at a, the transducer nondeterministically outputs ϵ or the root type corresponding to $tighten(a, d, B_{X_1}(\vec{X}))$. This proceeds inductively along the path to X, until L_X is computed. More precisely, for each type a in the alphabet of L_{X_i}, the language $L^a_{X_{i+1}}$ corresponding to the nodes X_{i+1} reachable from a by a vertical path p_{i+1} and a horizontal path $l_{i+1}.X_{i+1}.r_{i+1}$ is computed as above. The language $L_{X_{i+1}}$ is $\tau(L_{X_i})$ where τ is the substitution defined by $\tau(a) = L^a_{X_{i+1}}$ for each a in the alphabet of L_{X_i}. This is iterated to compute L_{X_j}, $j < k$, and finally L_X.

The development in this section leads to following key result:

Theorem 4.5 *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.[8] It remains open whether the complexity is tight.

Remark 4.6 *In this section we assumed that input lotos are described by regular ltds. Now*

[8] This complexity is to be expected: even very simple static analysis of queries is typically exponential. For example, recall that containment of conjunctive queries is NP-complete.

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 spacialized regular ltd. Our inference algorithm and Theorem 4.5 generalize easily to such input ltds.

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 4.7 *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 and the decidability of whether a context-free language is included in a regular language. We note that a different approach consisting of driving the translation process by the target ltd is considered in [MZ98].

Special cases

We consider ltd inference in several special cases of practical interest. 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 significant special cases 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.

First, we consider a natural restriction on the input ltds. 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. By revisiting the inference algorithm in the previous section in the case of stratified ltds, we are able to show the following.

Theorem 4.8 *Given a stratified regular ltd d and a selection loto-ql query q, 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.

Next, consider the special class of selection loto-ql queries which do not use recursive vertical navigation. That is, all 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*. We can show the following, analogous to the above.

Theorem 4.9 *Given a regular ltd d and a vertically nonrecursive loto-ql query q, one can effectively construct a tight specialized regular ltd for $q(T(d))$.*

Once again, the complexity remains 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 4.10 *(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.*

5 Extensions

In this section we consider ltd inference for several extensions of the model and query language. We begin by looking at general loto-ql queries (not restricted to selection queries), which construct new lotos using a powerful group-by construct defining nested lists. Then we briefly discuss other extensions.

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

construct $H(\vec{X})$
where $B(\vec{X})$

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 $type(X)$ (denoting the type of $root(X)$). The set of terms using variables from the body $B(\vec{X})$ is denoted $Terms(B(\vec{X}))$. The root is always labeled by a symbol. Internal nodes can only be labeled by a symbol or by a term $type(X)$. 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 $T_1, T_2, \ldots T_k$. We call a tree as above *parameterized* (by the terms it contains), and make the parameters explicit by writing $t(T_1, \ldots, T_k)$.

We next describe group-by labels. Each group-by label is a (possibly empty) sequence of distinct terms in $Terms(B(\vec{X}))$. Group-by labels are denoted $[T_1 \ldots T_k]$. 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;[9], 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.

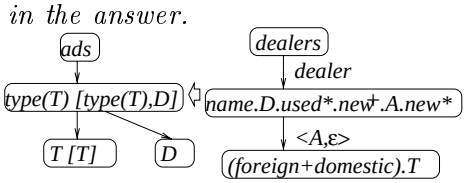
Given a query with body $B(\vec{X})$ and head $H(\vec{X})$, the answer to the query on given input is constructed from the set of bindings $\mathcal{B}$ of variables satisfying the pattern $B(\vec{X})$ (see definition of binding in Section 3). Each binding $\beta \in \mathcal{B}$ extends to terms $type(X)$ in the obvious way: if t is an input loto with labeling λ and β is a binding for the variables, then $\beta(type(X)) = \lambda(root(\beta(X)))$. The answer to the query is a loto constructed by performing a breadth-first traversal of $H(\vec{X})$ and replacing each subtree $t(T_1 \ldots T_k)$ whose root has group-by label $[T_1 \ldots T_k]$ by a list of subtrees $t(\beta(T_1) \ldots \beta(T_k))$, one for every binding $\beta \in \Pi_{T_1,\ldots,T_k}(\mathcal{B})$. The order of the subtrees in the list is given by the lexicographic order of the bindings (for terms of the form $type(X)$, 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 5.1 *Consider a "dealers" input containing car ads, partially described by the following ltd piece:*

[9] Empty labels are omitted in examples.

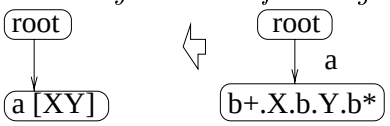
$root : dealers$;
$dealers : dealer^*$; $dealer : name, used^*, new^*$;
$used : (foreign + domestic + sedans + RVs)$;
$new : (foreign + domestic + sedans + RVs)$;
$foreign : model, (year + \epsilon)$;
$domestic : model, (year + \epsilon)$;

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 type(T) can only be "domestic" or "foreign"). In particular, there is one "domes tic" loto for each dealer who sells at least one domestic car and similarly for "foreign". Each "domestic" loto contains a list of all "domestic" ads published by the specific deale r. 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.

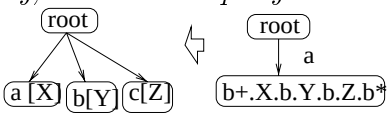


Ltd inference for general loto-ql queries The ltd inference algorithm we described for selection queries can be extended to general loto-ql queries. However, the ltd it produces is sound but no longer tight for the view. The sound ltd it produces can still be used in a variety of ways. For example, it 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 5.2 *The following query always produces in the answer lists of length n^2 which cannot be described by a context-free language.*



Similarly, consider the query



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 5.2 (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 $\overline{L}_{\vec{Z}}$ containing $L_{\vec{Z}}$. The language $\overline{L}_{\vec{Z}}$ 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 $\overline{L}_{\vec{Z}}$ is obtained using an appropriate sequence of substitutions. For instance, if $k = 2$, the language $\overline{L}_{Z_1Z_2}$ 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 $\overline{L}_{\vec{Z}}$ explicit, we denote the language $\overline{L}_{\vec{Z}}$ in context λ by $\overline{L}_{\vec{Z}}(\lambda)$.

Now suppose n is a symbol a. Then $L_n(\lambda)$ is contained in $h(\overline{L}_{\vec{Z}}(\lambda))$ where h is the homomorphism mapping every symbol of $\overline{L}_{\vec{Z}}(\lambda)$ to a. Next, suppose n is a variable Z_i in $\vec{Z}$. Then $L_n(\lambda)$ is contained in $h(\overline{L}_{\vec{Z}}(\lambda))$ 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 $\overline{L}_{\vec{Z}}(\lambda)$. 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 5.2 (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.

Other Extensions The loto model and loto-ql language provide bare-bones abstractions of XML and query languages for semistructured data. Various features are left out, such as:

(i) availability of attribute values and text content, and their use in query selection conditions (e.g. `X.name = Joe`, `X.title=Y.title`, or `X.name like Papa*`)

(ii) additional ordering criteria to construct the answer to a query, (e.g. `order-by X.price`);

(iii) richer list manipulation primitives to construct the answer to a query (e.g. list reversal, concatenation, etc).

The effect of such features on ltd inference is fairly minor, so long as the ltds themselves are not extended with additional information. Consider (i). Since ltds do not capture attribute value information, satisfaction of an ltd by a loto is orthogonal to the attribute values. Consider a query that uses conditions on attribute values. The primary impact on ltd inference arises

from the fact that tree patterns that involve non-trivial conditions on attribute values can be satisfiable or unsatisfiable, but never valid. This can be accounted for with a modification to the algorithm. Similarly, the inferred ltd is tight only in a weaker sense[10]: every document in the view satisfies the ltd, every name structure allowed by the ltd is realized by *some* document in the view, but, of course, not all documents satisfying the ltd are in the view. [PV99] captures this intuition using the notion of structural class.

Next, consider (ii). The ordering of bindings by attribute values is generally orthogonal to their ordering in the input. Let us say that the answer contains a list of the bindings of X, ordered by $X.price$. In the ltd for the view, the L_X obtained by our inference algorithm in Section 4 must be replaced by

$$\{\pi(u) \mid u \in L_X, \pi \text{ is a permutation of the letters in } u\}.$$

Again, this can be done with a minor adjustment to the inference algorithm. Finally, consider (iii). Many list manipulations primitives such as list reversal can be dealt with using straightforward operations on CFLs.

References

[Abi97] S. Abiteboul. Querying semistructured data. In *Proc. ICDT Conf.*, 1997.

[AH88] S. Abiteboul and R. Hull. Data functions, datalog and negation. In *Proc. ACM SIGMOD*, pages 143–153, 1988.

[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.

[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.

[BGL+] C. Baru, A. Gupta, B. Ludaescher, , R. Marciano, Y. Papakonstantinou, P. Velikhov, and A. Yannakopoulos. Xml-based information mediation with vamp. `http://www.npaci.edu/DICE/Pubs/vamp-demo.ps`.

[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.

[CKS81] A.K. Chandra, D.C. Kozen, and L. Stockmeyer. Alternation. *J. of the ACM*, 28:114–133, 1981.

[CM90] M. Consens and A. Mendelzon. Graphlog: a visual formalism for real life recursion. In *Proc. ACM Symp. on Principles of Database Systems*, 1990.

[10] And there is no ltd that is tight in a stronger sense.

[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 Intractibilitiy: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[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.

[Klu80] A. Klug. Calculating constraints on relational expressions. *ACM Transaction on Database Systems*, 5:260–290, 1980.

[KP82] A. Klug and R. Price. Determining view dependencies using tableaux. *ACM Transaction on Database Systems*, 7:361–381, 1982.

[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. `http://www.w3.org/TR/1998/NOTE-XML-data`.

[MW] A. Mendelzon and P. Wood. Finding regular simple paths in graph databases. *SIAM J. Comp.*

[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.

[NUWC97] S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Representative objects: Concise representations of semistructured, hierarchical data. In *Proc. ICDE*, 1997.

[PV] Y. Papakonstantinou and V. Vianu. Dtd inference for views of XML data.

[PV99] Y. Papakonstantinou and P. Velikhov. Enhancing semistructured data mediators with document type definitions. In *Proc. ICDE Conf.*, 1999.

[QRS+95] D. Quass, A. Rajaraman, S. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Proc. DOOD*, pages 319–44, 1995.

[Suc] D. Suciu. An overview of semistructured data. To appear in SIGACT News.