INFORMATICA, 2013, Vol. 24, No. 4, 577–602

Heuristic Methods for Inference of XML Schemas: Lessons Learned and Open Issues

Irena MLÝNKOVÁ*, Martin NEČASKÝ
Department of Software Engineering, Faculty of Mathematics and Physics
Charles University in Prague, Czech Republic
e-mail: mlynkova@ksi.mff.cuni.cz, necasky@ksi.mff.cuni.cz

Received: June 2011; accepted: October 2012

Abstract. In this paper we focus on a specific class of XML schema inference approaches – so-called *heuristic approaches*. Contrary to *grammar-inferring* approaches, their result does not belong to any specific class of grammars and, hence, we cannot say anything about their features from the point of view of theory of languages. However, the heuristic approaches still form a wider and more popular set of approaches due to natural and user-friendly strategies. We describe a general framework of the inference algorithms and we show how its particular phases can be further enhanced and optimized to get more reasonable and realistic output. The aim of the paper is (1) to provide a general overview of the heuristic inference process and existing approaches, (2) to sum up the improvements and optimizations we have proposed so far in our research group, and (3) to discuss possible extensions and open problems which need to be solved. Hence, it enables the reader to get acquainted with the field fast.
Key words: XML Schema inference, regular-tree grammars, heuristics, integrity constraints.

1. Introduction

Without any doubt the XML (Bray *et al.*, 2008) is currently a de-facto standard for data representation. Its popularity is given by the fact that it is well-defined, easy-to-use and, at the same time, enough powerful. To enable users to specify own allowed structure of XML documents, so-called *XML schema*, the W3C[1] has proposed two languages – DTD (Bray *et al.*, 2008) and XML Schema (Thompson *et al.*, 2004; Biron and Malhotra, 2004). The former one is directly a part of XML specification and due to its simplicity it is one of the most popular formats for schema specification. The latter language was proposed later, in reaction to the lack of constructs of DTD. The key emphasis is put on simple types, object-oriented features (such as user-defined data types, inheritance, substitutability etc.) and reusability of parts of a schema or whole schemas.

On the other hand, statistical analyses of real-world XML data show that a significant portion of XML documents (52% Mignet *et al.* (2003) of randomly crawled or 7.4% Mlýnková *et al.* (2006) of semi-automatically collected) still have no schema at all. What

*Corresponding author.
[1] `http://www.w3.org/`.

is more, XML Schema definitions (XSDs) are used even less (only for 0.09% Mignet *et al.* (2003) of randomly crawled or 38% Mlýnková *et al.* (2006) of semi-automatically collected XML documents) and even if they are used, they often (in 85% of cases Bex *et al.* (2004)) define so-called *local tree grammars* (Murata *et al.*, 2005), i.e., grammars that can be defined using DTD as well.

Consequently a new research area of *automatic inference of an XML schema* has opened. The key aim is to create an XML schema for the given sample set of XML documents that is neither too general, nor too restrictive. It means that the set of document instances of the inferred schema is not too broad in comparison with the sample data but, also, it is not equivalent to it. Currently, there are several proposals of respective algorithms, but there is also still a space for further improvements. In particular, since according to the Gold's theorem (Gold, 1967) regular languages are not identifiable only from positive examples (i.e., sample XML documents expected to be valid against the resulting schema), the existing methods need to exploit either heuristics or a restriction to an *identifiable* subclass of regular languages.

Contributions. In this paper we focus on a specific class of XML schema inference approaches – so-called *heuristic approaches*. Contrary to *grammar-inferring* approaches, their result does not belong to any specific class of grammars and, hence, we cannot say anything about their features from the point of view of theory of languages. However, the heuristic approaches still form a wider and more popular set of approaches due to natural and user-friendly strategies. We describe a general framework of the inference algorithms and we show how its particular phases can be further enhanced and optimized to get more reasonable and realistic output. The aim of the paper is (1) to provide a general overview of the heuristic inference process and existing approaches, (2) to sum up the improvements and optimizations we have proposed so far in our research group, and (3) to discuss possible extensions and open problems which need to be solved. Hence, it enables the reader to get acquainted with the field fast.

Outline. The rest of the paper is structured as follows: In Section 2 we introduce a formal view of XML schemas. In Section 3 we first describe a general overview of typical phases of XML schema inference algorithm and then analyze particular phases from the point of view of current and well as our improvements. In Section 4 we discuss the remaining open issues to be solved in the area of XML schema inference in general. Finally, in Section 5 we conclude.

2. Formal View of XML Schema Languages

For the purpose of the following text we need a formal view of XML documents, XML schema languages and mutual validity. In general, we can divide the described schema languages into *grammar-based* (i.e., DTD Bray *et al.*, 2008, XML Schema Thompson *et al.*, 2004; Biron and Malhotra, 2004, RELAX NG Murata, 2002) and *pattern-based* (i.e., Jelliffe, 2001). The majority of current papers deal with basic and most common structural specification of XML data that can be expressed using the grammar-based languages. In

other words, they do not deal with advanced integrity constraints that can be expressed using Schematron or XML Schema assertions. So, if not stated otherwise, we consider the same set.

The grammar-based XML schema languages we consider in the first parts of our text can be further classified according to their expressive power. We borrow and slightly modify for our purposes the definitions from (Murata *et al.*, 2005). We represent an XML document as directed labeled tree $t = (V, E)$ called *XML tree* whose vertices from set V represent XML elements and attributes and edges from set E represent the hierarchical structure. We also consider a common formalization of an XML schema in a form of a *regular tree grammar* (*RTG*) $G = (N, T, S, P)$ having a set of non-terminals N, a set of terminals T, a set of start symbols S and a set of production rules P. Terminals of the grammar specify allowed elements (and attributes) in the XML document and the *production rules*, resp. *regular expressions* (*REs*) on their right hand sides, specify their allowed content. An *XML tree valid* against a given regular tree grammar is then an XML tree which can be constructed by the rewriting rules of the grammar. Another possibility to formalize XML schemas are classical *finite state automata* (*FSA*). It is a well known fact that production rules of regular grammars can be expressed as finite state automata and vice versa.

In Murata *et al.* (2005), various classes of RTGs that correspond to particular XML schema languages were introduced. In particular, we can define so-called *local-tree grammars* that correspond to DTD and *single-type tree grammars* that correspond to XML Schema. Note that RELAX NG corresponds to general regular tree grammars.

Definition 1. Let us have an RTG $G = (N, T, S, P)$. Two non-terminals $A, B \in N$ are *competing* with each other if there exist two production rules $A \to ar_1$ and $B \to ar_2$, where $a \in T$ and r_1, r_2 are REs over N.

Definition 2. A *local tree grammar* (*LTG*) is a RTG without competing non-terminals. A tree language is a *local tree language* if it is generated by a local tree grammar.

Definition 3. A *single-type tree grammar* (*STTG*) is a RTG such that

- for each production rule, non-terminals in its content model do not compete with each other, and
- start symbols do not compete with each other.

A tree language is a *single type tree language* if it is generated by a single type tree grammar.

3. General Framework of Heuristic Inference Approaches

The studied problem of XML schema inference can be described as follows: Being given an input set of XML trees $I = \{t_1, t_2, \ldots, t_n\}$, we search for an XML schema, i.e., an RTG $G_I = (N_I, T_I, P_I, S_I)$, such that $\forall i \in [1, n] : t_i$ is valid against G_I. In particular, we are

searching for G_I that is enough concise, precise and, at the same time, general. This requirement indicates, that the optimal result is hard to define and, in general, there may exist several solutions, i.e., a set of candidate RTGs $O = \{G_I^1, G_I^2, \ldots, G_I^m\}$, such that $\forall i \in [1, n], \forall j \in [1, m] : t_i$ is valid against G_I^j, whereas we are looking for the optimal $G_I^{opt} \in O$.

The problem of finding G_I^{opt} can be viewed as a special kind of optimization problem called *combinatorial optimization problem* (*COP*) (Barták, 1998). Since the space of candidate RTGs O is theoretically infinite, we usually exploit a kind of greedy search or other heuristics.

Most of the existing works use the same strategy consisting of the following phases to solve this problem:

- Phase I. Derivation of initial grammar;
- Phase II. Clustering of production rules of initial grammar;
- Phase III. Inference of REs;
- Phase IV. Refactorization;
- Phase V. Inference of simple data types;
- Phase VI. Inference of integrity constraints;
- Phase VII. Expressing the inferred items in the target XML schema language.

The current methods differ in involving/omitting selected phases and, in particular, strategies they apply on them. In the following sections we describe in detail the current approaches and the improvements we have proposed so far.

3.1. *Phase I. Derivation of Initial Grammar*

The first phase of the inference process is the same in all existing works: For each element node e in any of the XML trees in I and its child nodes $e_1, e_2, \ldots, e_k$ we construct a production $\vec{p}_e$ of the form $e \to lab(e)(e_1\ e_2\ \ldots\ e_k)$, where $lab(e)$ denotes the label of e. The production rules form so-called *initial grammar* (*IG*).

Example 1. An example of an XML document (a) and respective IG (b) is depicted in Fig. 1.

Note that for the sake of clarity we start non-terminals of IG with capital letters and terminals of IG with small letters.

3.2. *Phase II. Clustering of Production Rules of IG*

In the second phase the production rules of IG need to be clustered, since for each cluster a single RE is inferred in phase III (see Section 3.3). A typical strategy of the existing works is to cluster the production rules simply on the basis of the equivalence of left-hand sides.

Example 2. An example of clusters of IG in Fig. 1 (omitting duplicities and the production rule for pcdata) is depicted in Fig. 2.

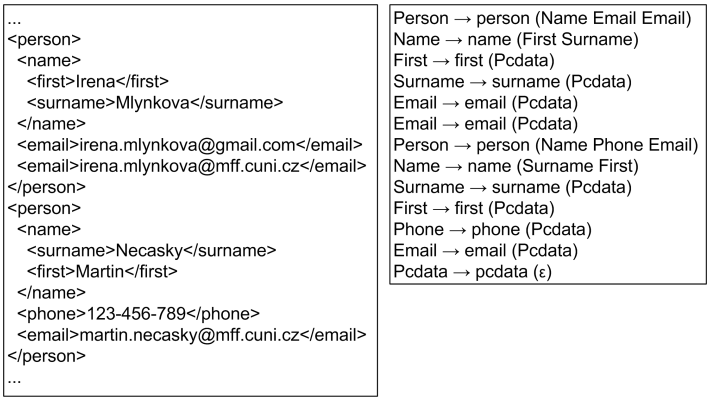

Fig. 1. An example of an XML document (a) and its IG (b).

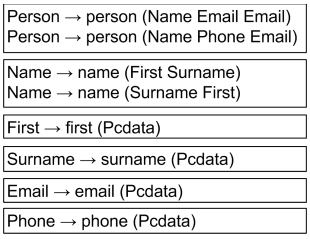

Fig. 2. Production rules of IG clustered according to equivalence of left hand sides.

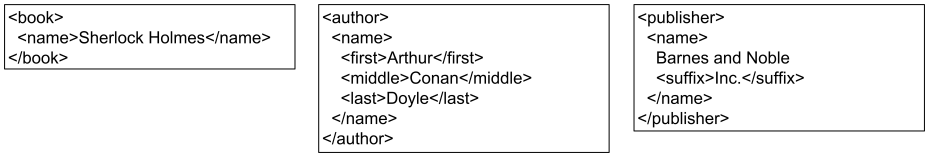

Fig. 3. Elements with the same name but different structure.

Apparently, such beginning step leads to inference of an LTG, i.e., DTD, where all the elements are defined at the same level, and, hence, we are not able to specify elements with the same name but different structure. However, the described functionality is included in XML Schema and RELAX NG, i.e., STTGs and RTGs, and can have quite reasonable usage due to homonymy of element names.

Example 3. Let us have an XML schema of a library where each book, author and publisher has a name. In the former case it can be only a simple string, whereas in the latter two cases the name can consist of a couple of elements each having its own semantics – see Fig. 3.

In paper (Vošta *et al.*, 2008) we have proposed two strategies that enable one to infer both STTGs and RTGs. They are based on two extensions of the clustering algorithm – on the basis of similarity of context of elements and similarity of content of elements. In paper Vyhnanovská and Mlýnková (2010) we have proposed another extension, that enables one to cluster also elements with different context (and names), but similar structure.

3.2.1. *Similarity of Context*

The key feature of STTGs is based on the fact that elements in the same content model, i.e., context, do not compete with each other. First, we need a formal definition of a context.

Definition 4. A *context* $cont_v$ of an element node $v \in V$ in a document tree $t = (V, E)$ is a concatenation of $lab(v_0)lab(v_1)\ldots lab(v_n)$, where v_0 is the root node of the tree, $v_n = v$, and $\forall i \in [1, n] : \langle v_{i-1}, v_i \rangle \in E$.

For the purpose of evaluation of similarity of two contexts $cont_v$ and $cont_u$, we exploit and modify the classical Levenshtein algorithm Levenshtein (1966) that determines the edit distance of two strings s_x, s_y using inserting, deleting or replacing a single character. In our case instead of single characters our operations work with whole element names. Hence, in the following text we can assume that we have a function $sim_{context} : V \times V \rightarrow [0, 1]$ which expresses the similarity of contexts of two element nodes, where 0 denotes strong dissimilarity and 1 equivalence.

3.2.2. *Similarity of Content*

The key feature of RTGs is that they allow for any kind of competing non-terminals. In other words, we can cluster the elements not only according to their context, but also content. As an XML element e can be viewed as a subtree t_e (in the following text denoted as an *element tree*) of corresponding document tree t, we use a modified idea of *tree edit distance*, where the similarity of trees t_e and t_f is expressed using the minimum number of edit operations necessary to transform t_e into t_f (or vice versa).

Currently, there exist several approaches to tree edit distance (e.g., Touzet, 2005; Marian, 2002). The key aspect is obviously the set of allowed edit operations which need to be suitable for a particular application.

Example 4. Consider two simple operations – adding and removal of a leaf node (and respective edge). As depicted in Fig. 4, such similarity is not suitable, e.g., for recursive elements. The example depicts two element trees of element a having subelement i having subelement j having subelement k which contains either subelement z or again i. With

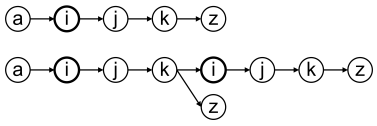

Fig. 4. Tree edit distance of recursive elements.

the two simple edit operations the edit distance would be 4, but, since the elements have the same XML schema, we would expect the optimal distance of 1 reflecting the usage of one additional recursive level.

For our purpose we exploit a similarity measure defined in Nierman and Jagadish (2002) which specifies more complex XML-aware tree edit operations on whole subtrees, each having its constant cost, as follows:

- *Insert* – a single node n is inserted to the position given by parent node p and ordinal number expressing its position among subelements of p;
- *Delete* – a leaf node n is deleted;
- *Relabel* – a node n is relabeled;
- *InsertTree* – a whole subtree t is inserted to the position given by parent node p and ordinal number expressing position of its root node among subelements of p;
- *DeleteTree* – a whole subtree t rooted at node n is deleted.

As it is obvious, for given trees t_e and t_f there are usually several possible transformation sequences for transforming t_e into t_f. A natural approach is to evaluate all the possibilities and to choose the one with the lowest cost. But such approach can be quite inefficient. Thus authors of Nierman and Jagadish (2002) propose so-called *allowable sequences* of edit operations, which significantly reduce the set of possibilities and, at the same time, speed up their cost evaluation.

In the following text we assume that we have a function $sim_{content} : V \times V \to [0, 1]$ which expresses the similarity of element trees t_e and t_f rooted at element nodes $e, f \in V$, i.e., content of elements.

3.2.3. *Clustering Algorithm*

In the resulting clustering algorithm we utilize a modification of classical *mutual neighborhood clustering* (*MNC*) *approach* (Jain and Dubes, 1988). We start with initial clusters $c_1, c_2, \ldots, c_k$ of elements given by the equivalence of their context, i.e., elements e and f belong to cluster c_i if $sim_{context}(e, f) = 1$. In other words, the initial clustering is based on a natural assumption that elements having the same context are likely to have the same schema definition. The initial clusters are then merged on the basis of element structure using the tree edit distance $sim_{content}$. Firstly, $\forall i \in [1, k]$ we determine a representative element r_i of cluster c_i.

3.3. *Phase III. Inference of REs*

Having the set of clusters $c_1, c_2, \ldots, c_l$ of production rules, the key emphasis is in the existing works put on inference of REs. A common approach is so-called *merging state algorithm* which consists of two steps:

1. $\forall i \in [1, l]$ a *prefix tree automaton* (*PTA*) A_{c_i} is built from the production rules of cluster c_i.
2. Each A_{c_i} is generalized via merging of its states.

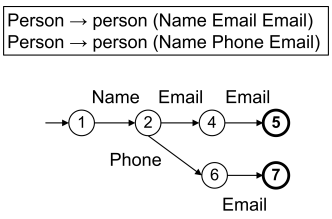

Fig. 5. An example of an IG and a PTA.

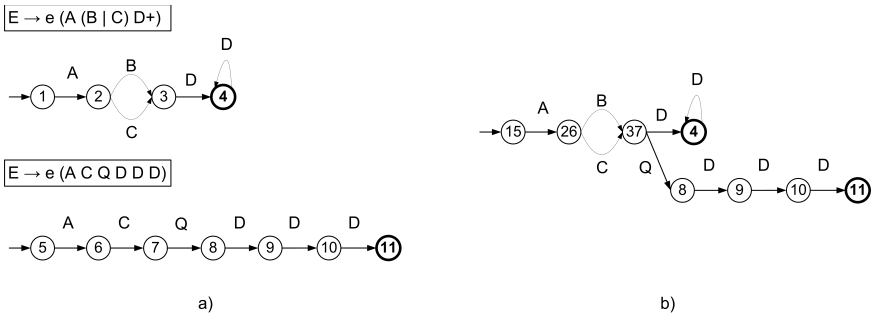

Fig. 6. Exploitation of an obsolete schema: (a) a production rule of an IG and a production rule of an obsolete schema, (b) result of their merging.

Example 5. An example of a cluster c_i of IG and the respective PTA A_{c_i} is depicted in Fig. 5.

In Mlýnková (2009) we have focussed on further extension of this step based on a typical real-world situation when a user creates an XML schema of XML documents but then modifies and updates only the data (identified in Mlýnková *et al.* (2006)). In other words, we are provided not only with the input documents $I = \{t_1, t_2, \ldots, t_n\}$, but also an *obsolete XML schema*, i.e., in general an RTG $G_{obs} = (N_{obs}, T_{obs}, P_{obs}, S_{obs})$, such that there exist parts of the input documents that are valid against it, i.e., the similarity of G_{obs} and G_I^{opt} is high. Such input can be exploited for speeding up the inference process. We only need to modify the process of creating the initial automata to be optimized and generalized. In particular, we cluster and merge the production rules of IG extracted from I together with production rules from P_{obs}. Hence, we do not start with PTAs, but general FSAs.

Example 6. An example of merging a production rule of IG with a production rule from an obsolete schema G_{obs} is depicted in Fig. 6.

Regardless the input automaton, the rules and strategies for merging the states of a PTA used in the current approaches differ, but they have a common aim to output a concise and precise XML schema. As we have mentioned at the beginning of this section, since the

$aaaa$	$\Rightarrow$	a^+
$(ab)^*\|a?b?$	$\Rightarrow$	$(a?b?)^*$
$ab\|ab^*$	$\Rightarrow$	ab^*
$a?, b?, c?$	$\Rightarrow$	$a\|b\|c$
a, b, c, d, a, d, b, c	$\Rightarrow$	$(a\|b\|c\|d)^+$

Fig. 7. Simple heuristic rules for merging states of automaton.

amount of possible output automata, i.e., schemas, is theoretically infinite, the approaches search only a subspace of possible solutions using a kind of greedy-search, terminating condition etc.

3.3.1. *Naive Solutions*
The first and simplest approaches implemented, e.g., in system *DTD-miner* (Moh *et al.*, 2000), use a simple set of heuristic rules such as those depicted in Fig. 7 and the search strategy continues until there exists a rule that can be applied.

3.3.2. *Evaluation of Schema Quality*
A strategy similar to *DTD-miner* is used also in system *XTRACT* (Garofalakis *et al.*, 2000). However, the rules are not applied using a greedy search, but a set of possible solutions is produced and the system selects the optimal one, i.e., it is able to evaluate quality of a schema generalization.

For the purpose of schema evaluation the authors (as well as the subsequent approaches) exploit so-called *minimum description length* (*MDL*) *principle* (Grunwald, 2005). It expresses the quality of an XML schema candidate using two aspects – conciseness and preciseness. *Conciseness* of an XML schema is expressed using the number of bits required to describe the schema itself (the smaller, the better). *Preciseness* of an XML schema is expressed using the number of bits required for description of the input XML trees in I using the schema. In other words, on the one hand, a good schema should be enough general which is related to the low number of states of the schema automaton, but, on the other hand, it should preserve details which means that it enables one to express document instances using short codes, since most of the information is carried by the schema itself.

3.3.3. *Advanced Merging Rules*
Apart from simple heuristic merging rules, there exist also approaches that base their strategy on various theoretical results.

The k, h*-context method* (Ahonen, 1996) specifies an identifiable subclass of regular languages which assumes that the context of elements is limited. So merging states of an automaton $A = (Q, \Sigma, \delta, s, F)$ is based on an assumption that two states $x, y \in Q$ are identical (and can be merged) if there exist two identical paths of length k terminating in x and y. In addition, also $h \leqslant k$ preceding states in these paths are then identical.

On the other hand, the s, k*-string method* (Wong and Sankey, 2003) is based on Nerod's equivalency of states of automaton A assuming that two states $x, y \in Q$ are equivalent if

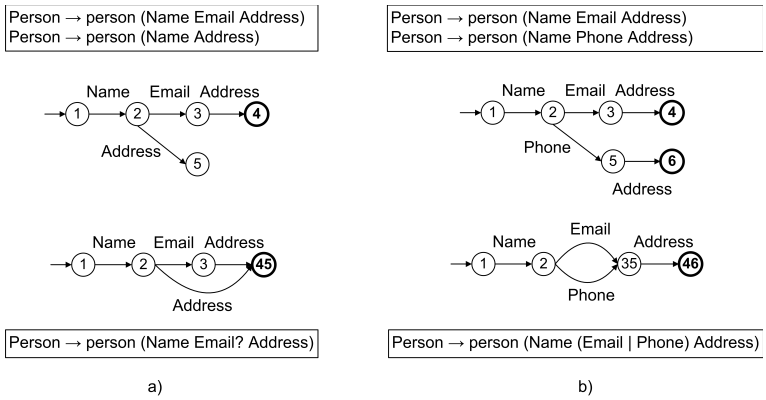

Fig. 8. Merging states of an automaton: (a) k, h-context and (b) s, k-string.

sets of all paths leading from x and y to any $f \in F$ are equivalent. But since such condition is hard to check, we can restrain to k-strings, i.e., only paths of length of k or terminating in a terminal state $f \in F$. The respective equivalency of states then depends on equivalency of sets of outgoing k-strings. In addition, for easier processing we can consider only s most probable paths, i.e., we can ignore singular special cases.

Example 7. Two examples of the described merging are depicted in Fig. 8. In the former case (a), we can merge states 4 and 5, because they have the same prefixes of length 1 (i.e., edge `Address`). In the latter case (b), we can merge states 3, 5 and 4, 6 because they have the same suffixes of length 1.

3.3.4. *Advanced Search Strategies*

As we have mentioned, we can view the problem of generalization of an automaton as a kind of optimization problem where we search for an optimal solution in a theoretically infinite space. The basic strategy is to use a kind of a greedy search. However, with the general view of the problem as a COP, we can use also more advanced meta-heuristics.

In Wong and Sankey (2003) the authors utilize a classical approach called *Ant Colony Optimization* (*ACO*) (Dorigo *et al.*, 2006). The ACO heuristic is based on observations of nature, in particular the way ants exchange information they have learnt. A set of artificial "ants" Δ search the space of solutions Θ trying to find the optimal solution $s_{opt} \in \Theta$ such that $\sigma(s_{opt}) \leqslant \sigma(s); \forall s \in \Theta$. In i-th iteration each ant $a \in \Delta$ searches a subspace of Θ for a local suboptimum until it "dies" after performing a predefined amount of steps N_{ant}. While searching, an ant a spreads a certain amount of "pheromone", i.e., a positive feedback which denotes how good solution it has found so far. This information is exploited by ants from the following iterations to choose better search steps. The key aspect of the algorithm is one step of an ant. Each step consists of generating of a set of possible movements, their evaluation using σ, and execution of one of the candidate steps. The executed step is selected randomly on the basis of probability given by σ. The algorithm terminates either after a specified number of iterations N_{iter} or if $s'_{opt} \in \Theta$ is reached such that $f(s'_{opt}) \leqslant$

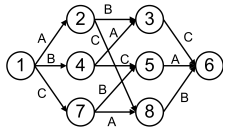

Fig. 9. Automaton $\mathcal{P}_3$ – permutation of three items.

$T_{\max}$, where $T_{\max}$ is a required threshold. Note that the randomness is the key aspect of the metaheuristic, since it enables one to search larger space of solutions than greedy strategies do and thus possibly find a better suboptimum.

In our case of schema inference we start with the PTA (or its modification) from phase II (see Section 3.2). A step of an ant means application of a selected generalization rule, either a naive one (see Section 3.3.1) or any of the advanced methods such as, e.g., k, h-context or s, k-string (see Section 3.3.3). The objective function σ can be defined, e.g., using the MDL principle (see Section 3.3.2).

3.3.5. *XML Schema Unordered Sequences*

In all the previous approaches the authors focussed on constructs that can be expressed in DTD. In Vošta *et al.* (2008) we extended the current approaches with the ability to infer also unordered sequences of elements, i.e., XML Schema `all` construct (or & operator for simplicity). (Even though, such construct can be expressed in DTD using a list all possible permutations, such approach is not used in practise for apparent disadvantages.) Our extension is based on the observation that it can be considered as a special kind of rule for merging states of an automaton. It enables one to replace a set of ordered sequences of elements with a single unordered sequence represented by the & operator.

In particular, we exploit the fact that for each $n \in \mathbb{N}$ we know the structure of the automaton $\mathcal{P}_n$ which accepts each permutation of n items having all the states fully merged.

Example 8. An example of automaton $\mathcal{P}_3$ is depicted in Fig. 9 for three items `A`, `B`, `C`.

Our idea is to find subautomata enough similar to $\mathcal{P}_n$ automaton. We can also observe, that the candidate subautomaton must be always a subgraph of $\mathcal{P}_n$, otherwise we can skip its processing. Hence, the problem of edit distance is highly simplified. and we use the following types of edit operations:

3.3.6. *XML Schema "Syntactic Sugar"*

As we have mentioned, XML Schema involves plenty of "syntactic sugar", i.e., constructs being *equivalent*. For instance, the unordered sequences described in the previous section represent one of the cases.

Definition 5. Let s_x and s_y be two XML schema fragments. Let $\eta(s) = \{d$ such that d is an element tree valid against $s\}$. Then s_x and s_y are *equivalent*, denoted as $s_x \sim s_y$, if $\eta(s_x) = \eta(s_y)$.

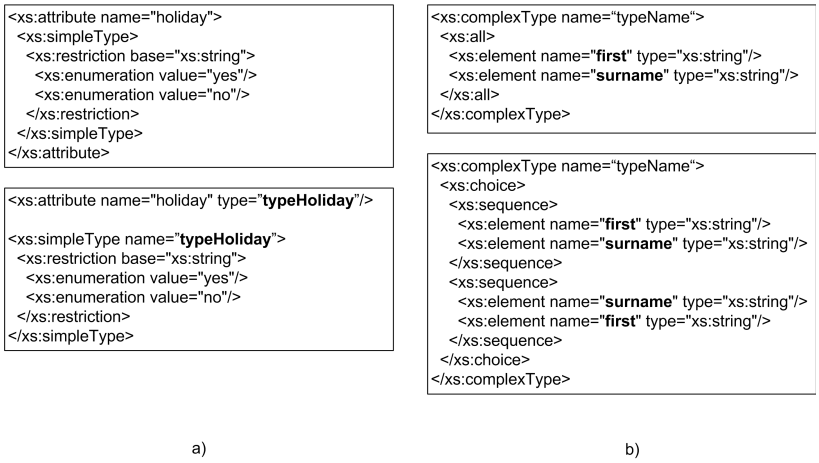

Fig. 10. "Syntactic sugar" of XML Schema: (a) globally and locally defined simple types, (b) unordered sequences.

Table 1
XSD equivalence classes of $\xi / \sim$.

Class	Constructs	Canonical representative
C_{ST}	Globally defined simple type, locally defined simple type	Locally defined simple type
C_{CT}	Globally defined complex type, locally defined complex type	Locally defined complex type
C_{El}	Referenced element, locally defined element	Locally defined element
C_{At}	Referenced attribute, locally defined attribute, attribute referenced via an attribute group	Locally defined attribute
C_{ElGr}	Content model referenced via an element group, locally defined content model	Locally defined content model
C_{Seq}	Unordered sequence of elements $e_1, e_2, \ldots, e_l$, choice of all possible ordered sequences of $e_1, e_2, \ldots, e_l$	Choice of all possible ordered sequences of $e_1, e_2, \ldots, e_l$
C_{CTDer}	Derived complex type, newly defined complex type	Newly defined complex type
C_{SubGr}	Elements in a substitution group γ, choice of elements in γ	Choice of elements in γ
C_{Sub}	Data types $\tau_1, \tau_2, \ldots, \tau_k$ derived from type τ, choice of content models defined in $\tau_1, \tau_2, \ldots, \tau_k, \tau$	Choice of content models defined in $\tau_1, \tau_2, \ldots, \tau_k, \tau$

Example 9. As depicted in Fig. 10(a), there is no difference if a simple type is defined locally or globally. Example (b) depicts the equivalence between an unordered sequence of a set of elements and a choice of their possible ordered permutations.

Consequently, having a set ξ of all XSD constructs, we can specify the quotient set $\xi / \sim$ of ξ by $\sim$ and respective equivalence classes Mlýnková and Nečaský (2009) – see Table 1. Each of the classes of $\sim$ equivalence can be then represented using a selected *canonical representative* as listed in Table 1 as well. Note that each of the constructs not mentioned in the table forms a single class $C_1, C_2, \ldots, C_n$.

With regard to the classes, we can generalize the previous approach for inference of unordered sequences, since from a general point of view we can find multiple ways how

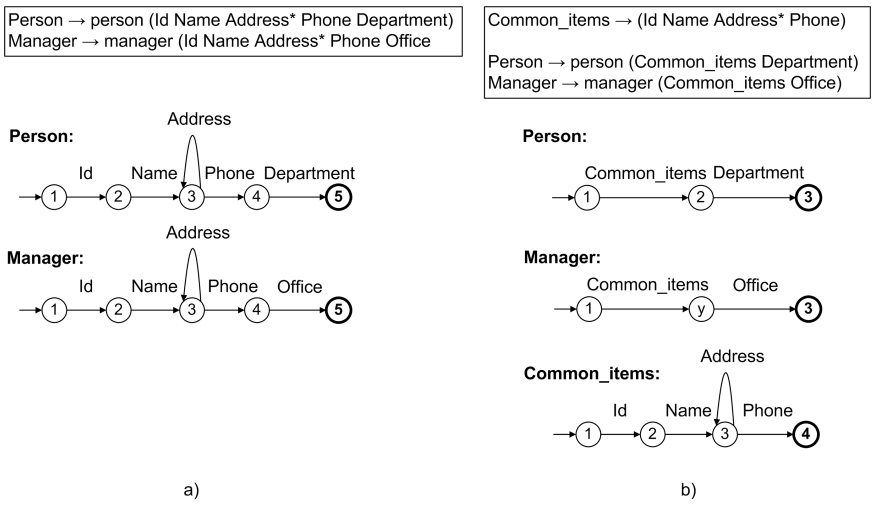

Fig. 11. Splitting and outlining an automaton: (a) productions with a common fragment, (b) outlined common fragment.

to express an XSD. For our purposes, we first slightly modify its definition (Vyhnanovská and Mlýnková, 2010).

The solution to the problem we proposed Mlýnková and Nečaský (2009) is taking into account also the semantics of the data. In particular, we use a kind of a thesaurus that enables one to discover semantic relations. Hence, we discover that the semantic similarity of words `person` and `book` is not high, but similarity of, e.g., `person`, `author` and `editor` is sufficient. And it also brings another advantage, since it indicates the way we should express the sharing (as discussed in Section 3.7).

Example 10. Let us have two candidates for outlining – `employee` and `manager`. The thesaurus not only determines that they are related, but also that `employee` is a *broader term* to `manager`. Hence, instead of creating `typeCommon` and derived types `typeEmployee` and `typeManager`, we create `typeEmployee` and a derived type `typeManager`.

To ensure the indicated functionality, we extend the process of inference of RE with an outlining rule. The algorithm directly searches for equivalent schema fragments and analyzes their semantic relevance. If it is identified as sufficient, the respective subautomaton is outlined.

Example 11. An example of outlining of an automaton is depicted in Figs. 11(a) and 11(b). In this case we exploit the fact that elements `person` and `manager` have common items that can be outlined into a globally defined schema fragment represented using a subautomaton. Since "person" and "manager" are semantically related, the sets are enough large and the items are not too general, the result again bears more information and it is more realistic.

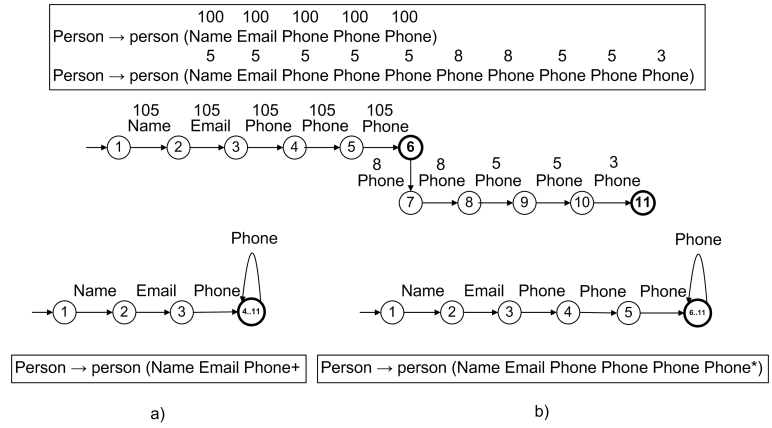

Fig. 12. Splitting repetitions: (a) application of merging rule $a, a, a, \ldots, a \Rightarrow a^+$, (b) application of splitting repetitions rule $a, a, a, \ldots, a \Rightarrow a, a, a, a^*$.

3.3.7. *Statistical Analysis of Input Documents*

If we want to go in the inference process even further and be more precise, we can exploit the given XML documents more deeply (Mlýnková and Nečaský, 2009). Our motivation results from ideas used in adaptive schema-driven *XML-to-relational mapping strategies*. They apply various XML-to-XML transformations (such as, e.g., $a^+ = a, a^*$) on the input XML schema, evaluate them and select the optimal one. For this purpose, we can exploit almost any equation known for regular expressions. However, since the amount of options is again large and we would get to the same problem as described above, we need to restrict ourselves to cases that can be assessed as relevant. But, since we do not have an etalon in a set of XML queries like the adaptive mapping methods do, we exploit etalons relevant to schema inference.

Example 12. For instance we can find out that in 95% of cases element `person` has two subelements `phone` at maximum and only in 5% of cases there occur elements `person` having more than five subelements `phone`. In the existing works the schema would be generalized to `phone+` or `phone*` although for most of the input XML documents it is too general. Consequently, if we exploit the above described equation we could preserve the first two occurrences of element `phone` and provide more realistic schema, i.e., `phone phone phone*`, bearing more precise information.

For the purpose of the indicated improvements we further extend the rules for merging states of an automaton. In particular, we add new merging rules that exploit statistics of the given XML data, i.e., while merging we take into account also additional information that influence the process.

Example 13. An example of exploitation of data statistics is depicted in Fig. 12. The numbers above the particular elements depict the amount of data instances that induce

$a??$	$\Rightarrow$	$a?$	$a^*?$	$\Rightarrow$	a^*	aa^*	$\Rightarrow$	a^+	$(ab)\|(ac)$	$\Rightarrow$	$a(b\|c)$
a^{++}	$\Rightarrow$	a^+	$a?^*$	$\Rightarrow$	a^*	a^+a^*	$\Rightarrow$	a^+			
a^{**}	$\Rightarrow$	a^*	a^{+*}	$\Rightarrow$	a^*	$a?a^+$	$\Rightarrow$	a^*			
a^{*+}	$\Rightarrow$	a^*									
$a?^+$	$\Rightarrow$	a^*									
$a^+?$	$\Rightarrow$	a^*									

Fig. 13. Merging of operators during refactorization.

this production rule. If we do not consider the statistics, we would infer Fig. 12(a). But, with regard to the data, it would be too general, since most of the `person` elements have right three subelements `phone`, whereas only in few cases there are persons with more phone numbers. Consequently, from the point of view of preciseness of information on the data, Fig. 12(b) is much realistic and bears more precise information.

3.4. *Phase IV. Refactorization*

A natural next phase of each schema inference method is *refactoring* or *refinement*, i.e., improving readability and simplifying structure while preserving the functionality of the resulting schema. Such requirement is ensured using a set of rules that enable one to describe equivalent or more general result. A demonstrative set of rules is depicted in Fig. 13.

As we can see, the specified rules enable one to remove duplicate occurrence operators, to merge sequences of distinct occurrence operators into a single one, to merge sequences of the same fragments, to avoid nondeterministic content models etc. Almost all of the mentioned approaches involve such a phase which indicates that the merging rules are not optimal and the approaches can still be improved. On the other hand, naturally, we can apply only those rules that do not interfere with other approaches, such as exploitation of splitting repetitions described in Section 3.3.7.

In phase III. (see Section 3.3) we have also discussed our extension (Mlýnková, 2009) of usage an obsolete schema. Apparently, using this approach, there may occur schema fragments that are not used in any of the input data. In other words, the input XML trees in I are valid against G_I^{opt}, however, it is too general and may cover also too distinct data. Hence, we extend the refactorization process with the following steps:

1. Pruning of unused schema fragments;
2. Correction of lower and upper bounds of occurrences of schema fragments;
3. Correction of operators.

All the three steps can be done using a single linear passing of the input documents in I and preserving respective flags for particular schema parts.

3.5. *Phase V. Inference of Simple Data Types*

Since most of the current approaches focus on inference of DTDs, they treat all text values as simple strings, i.e., they use common `PCDATA` data type. However, in XML Schema, as well as RELAX NG, we have a hierarchy of simple data types and also an option to specify user-defined types. Surprisingly, the inference problem of simple data types is currently highly marginalized. There seem to exist only two exceptions (Chidlovskii, 2002; Hegewald *et al.*, 2006) which utilize the same approach: Each set of values of an element/attribute is simply analyzed to identify the minimal data type which contains all of them. Nevertheless, the authors focus only on numeric data types (such as `decimal`, `float`, `long`, `negativeInteger`), `date`, `binary` and `string`. Broader sets of simple types or even user-defined simple types are not supported so far.

3.6. *Phase VI. Inference of Integrity Constraints*

Similarly to the case of inference of simple data types, the process of inference of *integrity constraints (ICs)* can be considered as an additional phase that can extend any of the inference strategies in general. An IC is a condition specified on the XML data. From this point of view we can consider simple data types as ICs as well. However, in this section we mean the "classical" ICs, in particular those that can be expressed in current XML schema languages. The most common type of ICs are keys and feign keys. In DTD they are expressed using simple data types `ID` and `IDREF(S)` and are valid in the context of the whole document. In XML Schema we are provided with constructs `unique`, `key`, and `keyref` which enable one to specify the context/scope of the constraint. Apart from that, XML Schema involves new constructs `assert` and `report` that correspond to Schematron rules. However, none of the current approaches focuses on these advanced constructs.

Basic foundations and classifications of XML keys and discussion of the related decision problems can be found in Buneman *et al.* (2003).

Definition 6. A *key* is a construct $(C, P, \{L_1, L_2, \ldots, L_k\})$, where $C, P, L_1, L_2, \ldots, L_k$ are XPath paths without predicates that use only child and descendant axes. C is called *context path*, P *target path* and $L_1, L_2, \ldots, L_k$ *key paths*. C can be omitted, i.e., we can write $(P, \{L_1, L_2, \ldots, L_k\})$. This is equivalent to $(/, P, \{L_1, L_2, \ldots, L_k\})$. If C is omitted we call the key *global key*, otherwise, it is called *relative key*.

For the sake of simplicity we can consider that $k = 1$, i.e., a key is a construct $(C, P, \{L\})$.

Example 14. A key (`//project`, `member`, {`mid`}) formalizes the XML Schema `key` example depicted in Fig. 14.

Since the authors of Buneman *et al.* (2003) do not provide a formalism for foreign keys, we extend it in a similar manner.

```
<element name="project">
 <!-- ... -->

 <key name="MemberKey">
  <selector path="member"/>
  <field path="mid"/>
 </key>

 <keyref name="MemberKeyRef" refer="MemberKey">
  <selector path="document"/>
  <field path="mid"/>
 </keyref>
</element>
```

Fig. 14. An example of XML Schema `key` and `keyref` constructs.

DEFINITION 7. A *foreign key* is a construct $(C, (P^1, \{L_1^1, L_2^1, \ldots, L_k^1\}) \Rightarrow (P^2, \{L_1^2, L_2^2, \ldots, L_k^2\}))$, where $(C, P^2, \{L_1^2, L_2^2, \ldots, L_k^2\})$ is a key and $P^1, L_1^1, L_2^1, \ldots, L_k^1$ are XPath paths without predicates that use only child and descendant axes. C can be omitted as in the case of keys.

For the sake of simplicity we can again consider that $k = 1$, i.e., a foreign key is a construct $(C, (P^1, \{L^1\}) \Rightarrow (P^2, \{L^2\}))$.

EXAMPLE 15. A foreign key (`//project`, (`document`, {`mid`}) ⇒ (`member`, {`mid`})) formalizes the XML Schema foreign key (`keyref` construct) depicted in Fig. 14.

Probably the first approach that enables one to search for XML keys can be found in Grahne and Zhu (2002). The authors first show that the set of candidate keys, i.e., sets of values that fulfill the condition of uniqueness in a specific context (see Definition 6), is large and we need to select the optimal one. They propose an algorithm based on a classical data-mining technique called *Apriori* which enables one to mine all frequent item sets. For finding *minimal* keys, i.e., the set where no key is inferable from others the authors exploit a sound and complete set of inference rules proposed in (Buneman *et al.*, 2003), such as, e.g.,

$$\begin{aligned} &\big(C, (P, S)\big) \wedge C' \subseteq C \rightarrow \big(C', (P, S)\big), \\ &\big(C, (P, S)\big) \wedge P' \subseteq P \rightarrow \big(C, (P', S)\big). \end{aligned} \tag{1}$$

Another aim of the authors is to find so-called *approximate* keys, i.e., those valid in "almost" the whole XML document. For this purpose they introduce two concepts – *support* and *confidence* of key expression, i.e., measures of interestingness of a key expression from the point of view of the input data.

In Barbosa and Mendelzon (2003) the authors focus on the problem of inference of `ID` and also `IDREF(S)` attributes. In case of keys, they focus on the same issue, i.e., to identify the optimal key from the set of candidates, in this case using a greedy search strategy. Then, having a set of keys, the finding of foreign keys means just checking the condition specified by Definition 7.

```
01 for $e in //employee
02 return
03  <employee>
04   {$e/name}
05   {for $m in //member[mid=$e/eid]
06   return
07    <member>{$m/../code,$m/position}</member>}
08   {let $d := //document[mid=$e/eid]
09   return
10    <doccnt>{count($d)}</doccnt>
11    <docavgpages>{avg($d/pages)}</docavgpages>}
12  </employee>
```

Fig. 15. Query with repeating join pattern.

In Nečaský and Mlýnková (2009) we propose an approach which enables to discover keys and foreign keys more precisely using the analysis of XML queries, in particular join queries. Assume a query Q which joins a sequence of elements S_1 targeted by a path P_1 with a sequence of elements S_2 targeted by a path P_2 on a condition $L_1 = L_2$. It means that Q joins an element e_1 from S_1 with an element e_2 from S_2 if e_1/L_1 equals to e_2/L_2.

Example 16. An example of such query is depicted in Fig. 15. It joins a sequence of elements targeted by a path `//employee` with a list of elements targeted by a path `//member` on a condition `eid = mid` at line `05`, i.e., an `//employee` element e is joined with a `//member` element m if `eid` of e equals to `mid` of m.

Assume that each join is done via a key/foreign key pair. Hence, we can infer from Q that L_1 is a key for elements in S_1 or L_2 is a key for elements in S_2 and the other is a foreign key referencing the key. From our sample query in Fig. 15, we can therefore infer (`//employee`, {*eid*}) or (`//member`, {*mid*}). We can also infer the respective foreign key, i.e., (`//member`, {*mid*}) $\Rightarrow$ (`//employee`, {*eid*}) or (`//employee`, {*eid*}) $\Rightarrow$ (`//member`, {*mid*}), respectively.

The problem is how to decide which of L_1 and L_2 is the key and which is the foreign key. For this purpose we analyze the constructs used in the query (e.g., `for` vs. `let` clauses, aggregation functions such as `avg`, `min`, `max` or `sum`, function `count` etc.). On the basis of their usage we are able to output a list of scored keys and for each key a list foreign keys referencing the key. The score of a key can be negative or positive. A negative score means that the key is not specified by the XML documents while positive means that it is satisfied. The absolute value of the score means how sure we are about it. The method is supposed to be used in combination with any of the methods which analyze the data to optimize the search process.

3.7. *Phase VII. Expressing the Inferred Items in the Target Language*

Last but not least we need to express the inferred schema using constructs of the target XML schema language. In case of DTD constructs it is a quite straightforward process since the inferred REs only need to be directly rewritten into the respective syntax using

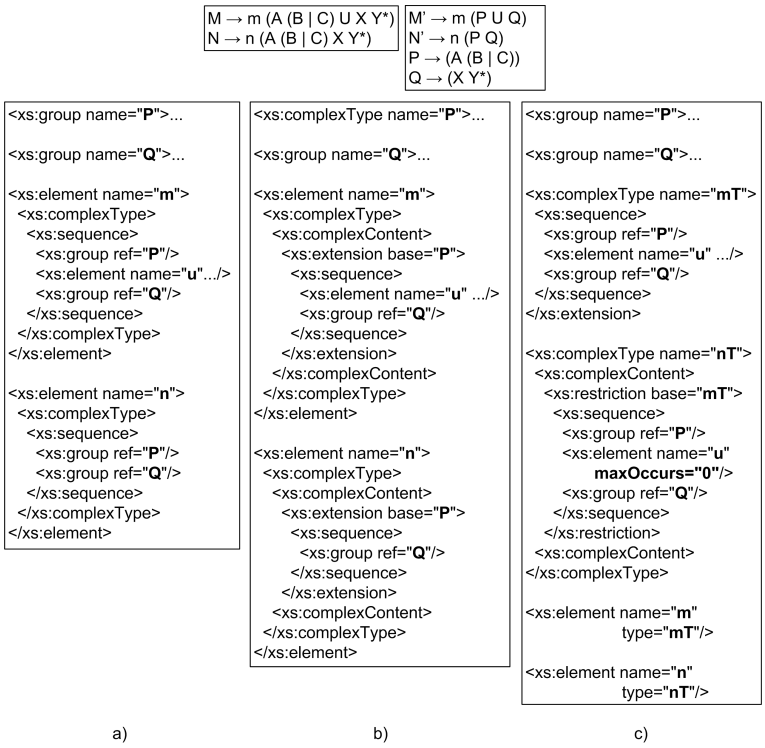

Fig. 16. Options of rewriting into XSD syntax.

classical approaches such as a *state removal method* (Linz, 2000) or an *algebraic method* Kain (1972). Note that a crucial issue of these methods is to select the best order of states to remove, since different removal sequences lead to different regular expressions. For this purpose several heuristics are proposed in Han and Wood (2007).

In case of XML Schema constructs the situation is much more complicated due to extensions we proposed (Vyhnanovská and Mlýnková, 2010; Mlýnková and Nečaský, 2009) and described in phase III (see Section 3.3). In general in all cases we can exploit classes C_{El}, C_{ElGr} and C_{At} (see Table 1). Depending on the type of the outlined production rule, we simply define respective globally defined element, group of elements, attribute or group of attributes. The problem is when we want use also classes C_{CT} and C_{CTDer}, i.e., globally defined complex types and their mutual derivation. Firstly, we naturally need to follow the W3C specifications (Thompson *et al.*, 2004). In particular, we have two choices – extension and restriction. Consequently, we can exploit outlining of complex types and their derivation only in case it can be specified in either of the two ways. Otherwise we must use its combination with classes C_{El}, C_{ElGr} and C_{At}. In addition, in general we usually have multiple options how to define the type hierarchy.

Example 17. Consider the example in Fig. 16, where $\vec{P}$ and $\vec{Q}$ denote production rules outlined from original production rules $\vec{M}$ and $\vec{N}$. In case (a) we exploit class C_{ElGr} and

define groups P and Q that are referenced from element definitions of m and n. In case (b) we combine classes C_{ElGr}, C_{CT} and C_{CTDer} and define complex type P to be the ancestor of complex types of both elements m and n. Similarly, in case (cNote that the authors of (Buneman *et al.*, 2003) define keys with more key paths.) we define content model of m, i.e., type mT, to be the ancestor of type nT derived by restriction.

For selecting the optimal choice we again exploit a thesaurus, but this time we do not determine semantic similarity of element names, but their mutual hierarchy. The rules are relatively simple:

- Relationships *Broader Term*/*Narrower Term*, which specify more/less general terms, determine the broader term to be the candidate for ancestor.
- Relationships *Use*/*Used For*, which specify authorized/unauthorized terms, determine any of the terms to be the candidate for ancestor.
- Relationship *Related Term* determines related terms, for whom a common ancestor is created.
- In other cases, i.e., when the terms are not related or we cannot say anything about their semantic relationship, we do not exploit derived complex types.

EXAMPLE 18. Consider again the example in Fig. 16. If we knew that "m" is broader term of "n" (e.g., "employee" and "director"), we would choose schema c). If we knew that "m" and "n" are related terms (e.g., "cat" and "dog"), we would choose schema (b). And if we knew that "m" and "n" are not related, but their content models are so similar that they were selected for merging, we would choose schema (a).

If we consider the resulting schema of existing approaches, i.e., two element definitions that involve separate local specifications of their content models, we can see that any of the three results is much more informative and precise.

3.8. *User Interaction*

Most of the previously described phases of general inference algorithm can be further optimized with the usage of user interaction (UI). The optimal situation would be if a user directly specified, e.g., the clusters of IG, the required merging rules, the outlined automata, target XML Schema constructs etc. However, since our primary aim is automatic inference, we cannot expect the user to execute the merging process, but to help it. Another problem is that the amount of user decisions cannot be too high, otherwise no one would exploit it.

In Vyhnanovská and Mlýnková (2010) we have proposed several cases when the inference process can benefit from user interaction. In particular we discuss the situations when the user can confirm/reject:

- clusters of IG to be merged, including clusters of elements with different names (phase II),
- subautomata to be outlined (phase III),

- candidates for unordered sequences (phase III), and
- XML Schema constructs (i.e., type inheritance and shared fragments) to be used in the target schema (phase VII).

Since all the cases are based on exploitation of a kind of similarity measure, we exploit a simple and straightforward two-threshold approach: If the particular similarity falls below $threshold_1$, the candidate is rejected automatically. If the distance falls between $threshold_1$ and $threshold_2$, UI is required. And if the result of similarity measure is higher than $threshold_2$, the candidate is automatically confirmed.

4. Open Issues

Although each of the existing approaches brings certain interesting ideas and optimizations, there is still a space of possible future improvements. We describe and discuss them in this section.

- **User Interaction**
  In most of the existing papers the approaches focus on purely automatic inference of an XML schema. The problem is that the resulting schema may be highly unnatural. Although, e.g., the MDL principle evaluates the quality of the schema using a realistic assumption that it should tightly represent the data and, at the same time, be concise and compact, user preferences can be quite different. (Note that this is not the same motivation as in case of papers Bex *et al*. (2006, 2007) that focus on real-world DTDs and XSDs.) Hence, a natural improvement may be exploitation of user interaction. Some of the existing papers (e.g., Ahonen, 1996) mention the aspect of user interaction, typically in phase IV (see Section 3.4) of refinement of the result, but there seems to be no detailed study and, in particular, respective implementation. And, naturally, this problem is closely related to a suitable user interface which does not require complex operations and decisions.
- **Other Input Information**
  In all the existing works the XML schema is inferred on the basis of a set of positive examples, i.e., XML documents that should be valid against the inferred schema. As we have mentioned, the Gold's theorem highly restricts the existing solutions and, hence, the authors focus on heuristic approaches or limit the methods to particular identifiable classes of languages. But another natural solution to the problem is to exploit additional information, such as an XML schema or XML queries. In addition, there seems to be no approach that would exploit negative examples (i.e., XML documents that should not conform to the schema). In this case we can find a real-world motivation again in the area of data evolution and versioning.
- **XML Schema Simple Data Types**
  One of the biggest advantages of the XML Schema language in comparison to DTD is its wide support of simple data types (Biron and Malhotra, 2004). It involves 44 built-in data types such as, e.g., `string`, `integer`, `date` etc., as well as user-defined data types derived from existing simple types using `simpleType` construct. In enables one to derive new data types using *restriction* of values of an existing type (e.g., a string value having length greater than two), *list* of values of an

existing type (e.g., list of integer values) or *union* of values of existing data types (e.g., union of positive and negative integers). Hence, a natural improvement of the existing approaches is a precise inference of simple data types. Unfortunately, most of the existing approaches omit the simple data types and consider all the values as strings. As we have mentioned, two exceptions are proposed in Chidlovskii (2002), Hegewald *et al.* (2006), but both the algorithms focus only on selected built-in data types.

- **XML Schema Advanced Constructs**
  The second big advantage of the XML Schema language are various complex constructs. The language exploits object-oriented features, such as user-defined data types, inheritance, polymorphism, i.e., substitutability of both data types and elements etc. Although most of these constructs do not extend the expressive power of XML Schema in comparison to DTD (Mlýnková, 2008), they enable one to specify more user-friendly and, hence, realistic schemas. Naturally, their usage is closely related to the previously described problem of user-interaction, since the user can specify which of the constructs are preferred.
- **Integrity Constraints**
  Both DTD and XML Schema enable one to specify not only the structure of the data, but also various semantic constraints. The current works focus mainly on the `ID`, `IDREF(S)` attributes (Grahne and Zhu, 2002; Barbosa and Mendelzon, 2003) and exploit various data mining approaches to find the optimal sets of keys and foreign keys. In Nečaský and Mlýnková (2009) we optimize the search strategy using analysis of XML queries. Unfortunately, all the existing works infer the keys separately, i.e., regardless a possibly existing XML schema or on the basis of an inference approach. Similarly, none of them focusses on any of the advanced constraints of XML Schema or Schematron. In addition, there are also more complex XML integrity constraints (Opočenská and Kopecký, 2008) that could be inferred, though they cannot be expressed in the existing schema specification languages so far, functional dependencies (Yu and Jagadish, 2008) or even languages for expressing any integrity constraint in general, such as, e.g., *Object Constraint Language (OCL)* (OMG, 2009). Their inference would extend the optimization of approaches that analyze and exploit information on XML data from XML schemas.
- **Other Schema Definition Languages**
  The DTD and XML Schema are naturally not the only languages for definition of structure of XML data, though they are undoubtedly the most popular ones. The obvious reason is that these two have been proposed by the W3C, whereas DTD is even a part of specification of XML. Nevertheless, there are also other relatively popular schema specification languages, the two most popular ones, RELAX NG and Schematron. The former language has higher expressive power than XML Schema and DTD, since it enables one, e.g., to combine elements and attributes in the regular expressions. The latter one exploits completely different approach (since it is a pattern-based, not grammar-based language) and, hence, it will require completely different inference approach.

5. Summary and Future Work

In general, the XML schema of XML documents is currently exploited mainly for two purposes – data-exchange and optimization. In the former case we usually need the inferred schema as a candidate schema further improved by a user using an appropriate editor, or in cases when no schema is available. In the latter case the approaches exploit the knowledge of the schema, i.e., the expected structure of the data, for optimization purposes such as, e.g., finding the optimal storage or compression strategy. However, in general, almost any approach that deals with XML data can benefit from the knowledge of their structure, i.e., XML schema. The only question is to what extent.

In this paper we focussed on the most popular group of approaches for semi-automatic inference of XML schema for a given set of XML documents – heuristic methods. Their popularity is given by the fact that their key aim is to provide natural and realistic results, despite we cannot specify any special features of the resulting schemas.

The main contribution of this paper can be summed up as follows:

- A general framework that characterizes the classical phases of algorithm for heuristic inference of XML schemas.
- A study of current approaches and their improvements of the particular phases of the general inference algorithm.
- A detailed description of several optimization approaches we proposed in recent years and their comparison with current approaches.
- A study of open issues to be solved in the area of schema inference in general.

Recently we have finished implementation of a general and extensible framework called *jInfer* (Klempa *et al.*, 2012a) that covers the general inference phases. Using plugins it enables one to change the respective approaches and compare their influence on the inference process as well as results in general. (Note that a similar system, called *SchemaScope*, focussing on the grammar-inferring approaches, was described in Bex *et al.* (2008).) In the next phase we will implement the current approaches and provide their detailed analysis and comparison using nontrivial set of both real-world and synthetic data. And, finally, in our future work we will focus mainly on the open issues stated in Section 4. Our primary aim is to study the advantages of other input information (such as XML queries or XML operations in general) and to infer broader information on the data, in particular integrity constraints. Our preliminary results in this area using *jInfer* can be found in Švirec and Mlýnková, 2012) ((2012)), Klempa *et al.* (2012b), Vitásek and Mlýnková (2012).

Acknowledgement. This work was partially supported by the Czech Science Foundation (GAČR), grant number P202/10/0573.

References

Ahonen, H. (1996). *Generating Grammars for Structured Documents Using Grammatical Inference Methods*. Report A-1996-4, Department of Computer Science, University of Helsinki.

Barbosa, D., Mendelzon, A. (2003). Finding ID attributes in XML documents. In: *Proceedings of Xsym 2003*. *Lecture Notes in Computer Science*, vol. 2824. Springer, Berlin, pp. 180–194.

Barták, R. (1998). *On-Line Guide to Constraint Programming*. http://kti.mff.cuni.cz/ bartak/constraints/.

Bex, G.J., Neven, F., Van den Bussche, J. (2004). DTDs versus XML schema: a practical study. In: *Proceedings of WebDB '04*. ACM Press, New York, pp. 79–84.

Bex, G.J., Neven, F., Schwentick, T., Tuyls, K. (2006). Inference of concise DTDs from XML data. In: *Proceedings of VLDB '06*, Seoul, pp. 115–126.

Bex, G.J., Neven, F., Vansummeren, S. (2007). Inferring XML schema definitions from XML Data. In: *Proceedings of VLDB '07*, Vienna, pp. 998–1009.

Bex, G.J., Neven, F., Vansummeren, S. (2008). SchemaScope: a system for inferring and cleaning XML schemas. In: *Proceedings of SIGMOD '08*, Vancouver, pp. 1259–1262. ACM Press, New York.

Biron, P.V., Malhotra, A. (2004). *XML Schema. Part 2. Datatypes*, 2nd ed. W3C. http://www.w3.org/TR/xmlschema-2/.

Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F. (2008). *Extensible Markup Language (XML) 1.0*, 5th ed. W3C. http://www.w3.org/TR/REC-xml.

Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.-C. (2003). Reasoning about keys for XML. *Information Systems*, 28(8), 1037–1063.

Chidlovskii, B. (2002). Schema extraction from XML collections. In: *Proceedings of JCDL '02*, Portland, pp. 291–292. ACM Press, New York.

Dorigo, M., Birattari, M., Stutzle, T. (2006). *Ant colony optimization – artificial ants as a computational intelligence technique*. Report TR/IRIDIA/2006-023, IRIDIA, Bruxelles, Belgium.

Garofalakis, M., Gionis, A., Rastogi, R., Seshadri, S., Shim, K. (2000). XTRACT: a system for extracting document type descriptors from XML documents. *SIGMOD Record*, 29(2), 165–176.

Gold, E.M. (1967). Language identification in the limit. *Information and Control*, 10(5), 447–474.

Grahne, G., Zhu, J. (2002). Discovering approximate keys in XML data. In: *Proceedings of CIKM '02*, McLean, pp. 453–460. ACM Press, New York.

Grunwald, P.D. (2005). *A Tutorial Introduction to the Minimum Description Principle*. Centrum voor Wiskunde en Informatica. http://homepages.cwi.nl/ pdg/ftp/mdlintro.pdf.

Han, Y.-O., Wood, D. (2007). Obtaining shorter regular expressions from finite-state automata. *Theoretical Computer Science*, 370(1–3), 110–120.

Hegewald, J., Naumann, F., Weis, M. (2006). XStruct: efficient schema extraction from multiple and large XML documents. In: *Proceedings of ICDEW '06*. IEEE Computer Society, Washington, pp. 81–81.

Jain, A.K., Dubes, R.C. (1988). *Algorithms for Clustering Data*. Prentice-Hall, Upper Saddle River.

Jelliffe, R. (2001). *The Schematron – An XML Structure Validation Language using Patterns in Trees*. http://xml.ascc.net/resource/schematron/.

Kain, R.Y. (1972). *Automata Theory: Machines and Languages*. McGraw-Hill Inc., New York.

Klempa, M., Mikula, M., Smetana, R., Švirec, M., Vitásek, M. (2012a). *jInfer XML Schema Inference Framework*. http://jinfer.sourceforge.net/modules/paper.pdf.

Klempa, M., Stárka, J., Mlýnková, I. (2012b). Optimization and refinement of XML schema inference approaches. In: *Proceedings of ANT '12*, vol. 10, Niagara Falls, pp. 120–127. Elsevier, Amsderdam

Levenshtein, V.I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8), 707–710.

Linz, P. (2000). *An Introduction to Formal Languages and Automata*. Jones & Bartlett Publishers, 3rd ed.

Marian, A. (2002). Detecting changes in XML documents. In: *Proceedings of ICDE'02*. IEEE Computer Society, Washington, pp. 41–52.

Mignet, L., Barbosa, D., Veltri, P. (2003). The XML web: a first study. In: *Proceedings of WWW'03*, Budapest, pp. 500–510. ACM Press, New York.

Mlýnková, I. (2009). On inference of XML schema with the knowledge of an obsolete one. In: *Proceedings of ADC '09*. Australian Computer Society, Inc., Darlinghurst, pp. 77–84.

Mlýnková, I., Nečaský, M. (2009). Towards inference of more realistic XSDs. In: *Proceedings of SAC '09*, Honolulu, pp. 639–646. ACM Press, New York.

Mlýnková, I., Toman, K., Pokorný, J. (2006). Statistical analysis of real XML data collections. *Proceedings of COMAD'06*. Tata McGraw-Hill, New Delhi, pp. 20–31.

Mlýnková, I. (2008b). Similarity of XML schema definitions. *Proceeding of DocEng '08*, Sao Paulo, pp. 187–190. ACM Press, New York.

Moh, C.-H., Lim, E.-P., Ng, W.-K. (2000). Re-engineering structures from web documents. In: *Proceedings of DL '00*, San Antonio, pp. 67–76. ACM Press, New York.

Murata, M. (2002). *RELAX (Regular Language Description for XML)*. `http://www.xml.gr.jp/relax/`.

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

Nečaský, M., Mlýnková, I. (2009). Discovering XML keys and foreign keys in queries. In: *Proceedings of SAC '09*, Honolulu, pp. 632–638. ACM Press, New York.

Nierman, A., Jagadish, H.V. (2002). Evaluating structural similarity in XML documents. In: *Proceedings of WebDB'02*, Madison, pp. 61–66. ACM Press, New York.

OMG (2009). *Object Constraint Language Specification, Version 2.0*. `http://www.omg.org/technology/documents/formal/ocl.htm`.

Opočenská, K., Kopecký, M. (2008). Incox – a language for XML integrity constraints description. In: *Proceedings of DATESO'08*. CEUR-WS.org, Desna – Cerna Ricka, Czech Republic, pp. 1–12.

Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N. (2004). *XML Schema. Part 1. Structures*, 2nd ed. W3C. `http://www.w3.org/TR/xmlschema-1/`.

Touzet, H. (2005). A linear tree edit distance algorithm for similar ordered trees. In: *Proceedings of CPM '05*. Lecture Notes in Computer Science. Springer, Heidelberg, pp. 334–345.

Vitásek, M., Mlýnková, I. (2012, in press). Inference of XML integrity constraints. In: *Proceedings of ADBIS '12*, Poznan. Springer, Berlin.

Vošta, O., Mlýnková, I., Pokorný, J. (2008). Even an ant can create an XSD. In: *Proceedings of DASFAA'08*, New Delhi, pp. 35–50. Springer, Berlin.

Švirec, M., Mlýnková, I. (2012). Efficient detection of XML integrity constraints violation. In: *Proceedings of NDT '12*, Dubai, UAE, pp. 259–273. Springer, Berlin.

Vyhnanovská, J., Mlýnková, I. (2010). Interactive inference of XML schemas. In: *Proceedings of RCIS '10*, Nice, pp. 191–202. IEEE Computer Society, Los Alamitos.

Wong, R.K., Sankey, J. (2003). *On Structural Inference for XML Data*. Report UNSW-CSE-TR-0313, School of Computer Science, The University of New South Wales.

Yu, C., Jagadish, H.V. (2008). XML schema refinement through redundancy detection and normalization. *The VLDB Journal*, 17(2), 203–223.

I. Mlýnková received her PhD degree in Computer Science in 2007 from the Charles University in Prague, Czech Republic. She is an assistant professor at the Department of Software Engineering of the Charles University and an external member of the Department of Computer Science and Engineering of the Czech Technical University. She has published more than 60 publications, 4 gained the Best Paper Award. She is a PC member or reviewer of 15 international events and co-organizer of 3 international workshops (X-Schemas@ADBIS, MoViX@DEXA, BenchmarX@DASFAA, all since 2009).

M. Nečaský received his PhD degree in Computer Science in 2008 from the Charles University in Prague, Czech Republic, where he currently works in the Department of Software Engineering as an assistant professor. He is an external member of the Department of Computer Science and Engineering of the Faculty of Electrical Engineering, Czech Technical University in Prague. His research areas involve XML data design, integration and evolution. He is an organizer or PC chair of three international workshops. He has published 15 refereed conference papers (2 received the Best Paper Award). He has published 3 book chapters and a book.

Euristiniai XML schemų išvedimo metodai: išmoktos pamokos ir neišspręstos problemos

Irena MLÝNKOVÁ, Martin NEČASKÝ

Pagrindinis dėmesys straipsnyje skirtas specifinei XML schemų išvedimų klasei, euristiniams išvedimams. Skirtingai nuo gramatinių išvedimų, euristiniais išvedimais gautos schemos negali būti aprašytos jokia konkrečia gramatika ir dėl to nieko negalima pasakyti apie jų savybes apibrėžiamas kalbų teorija. Nepaisant šio fakto, euristinių išvedimų klasė yra plati ir populiari, nes žmogui tokių išvedimų strategijos yra natūralios ir patogios. Straipsnyje aprašomos bendrosios euristinių algoritmų savybės ir parodoma kaip, norint gauti labiau pagrįstus ir realistiškesnius rezultatus, galima patobulinti ir optimizuoti atskirus tokių algoritmų žingsnius. Šiuo straipsniu siekta: (1) apžvelgti euristinio išvedimo procesą ir skirtingas jo traktuotes; (2) apibendrinti mūsų mokslinės grupės pateiktus pasiūlymus kaip tobulinti ir optimizuoti euristinio išvedimo procesą; (3) aptarti neišspręstas problemas ir galimas mūsų rezultatų plėtros kryptis. Straipsnis turėtų skaitytojui padėti greitai susipažinti su mūsų nagrinėjama tyrimų sritimi.