

Sample-based Decision Support for Specifying XML Constraints

Sven Hartmann

Department of Informatics, Clausthal University of Technology, Germany
Email: sven.hartmann@tu-clausthal.de

Sebastian Link

School of Information Management, Victoria University of Wellington, New Zealand
Email: sebastian.link@vuw.ac.nz

Thu Trinh

Department of Informatics, Clausthal University of Technology, Germany
Email: thu.trinh@tu-clausthal.de

Constraints can express important semantic information about the target XML repository. This information is utilised to model, store and process documents appropriately and more efficiently. In practice, it is challenging to single out the conditions that best capture the semantics of the application domain. Consequently, the task of identifying all relevant constraints is both crucial and difficult.

We show how sample documents assist participants of the design process in making informed choices about the specification of Boolean constraints on XML documents. Indeed, the decision whether such a constraint should be specified explicitly is reduced to the problem whether there is any document in our XML sample collection that is relevant for the underlying application domain. Furthermore, we establish how off-the-shelf tools for solving problems in propositional logic can be used to generate the collection of our sample documents semi-automatically.

ACM Classification: H.2.1. (Logical Design) Data Models, Schema and Subschemas; I.2.6. (Learning) Knowledge Acquisition; F.4.1. (Mathematical Logic) Mechanical Theorem Proving

1. INTRODUCTION

The design of XML repositories is based on the assumption that the semantics of the underlying application domain has been completely captured by the present model. Therefore, the acquisition of domain knowledge is absolutely crucial for the quality of the future XML depot (Bray *et al.*, 2006).

Integrity constraints restrict the class of future XML documents to those which are considered meaningful for the application at hand. However, the gathering of integrity constraints demands high abstraction abilities and an advanced understanding of logics. In practice, several people participate in the requirements process and, hence, may interpret the same constraint in different ways. Considering these problems and the significance of acquiring the complete semantic knowledge any assistance in the decision process is welcome.

Naturally, constraints are accrued by inspecting reasonable candidates one by one: a user or administrator suggests that some candidate φ captures significant properties of the target XML

Copyright© 2010, Australian Computer Society Inc. General permission to republish, but not for profit, all or part of this material is granted, provided that the JRPIT copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society Inc.

Manuscript received: 30 May 2008
Communicating Editor: Kate Smith-Miles

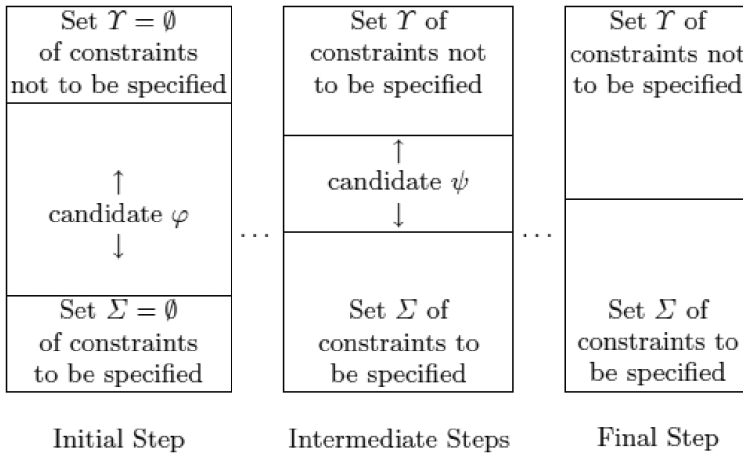


Figure 1: Specifying and Discarding of Candidate Constraints

repository, and according to the mutual feeling of all design participants φ will be added to the set Σ of constraints that have already been specified explicitly or φ will be discarded, i.e., added to the set Υ . Clearly, whenever the candidate φ is implied by Σ , then there is no need to specify it explicitly anymore. However, it may not be obvious at all whether φ is already specified implicitly. At the end of the acquisition process all potential candidates have been considered and the set Σ contains all those constraints that will be specified explicitly. This process is illustrated in Figure 1. The crucial question is how to deal with those candidates φ that are not implied by the current Σ ?

We will argue that XML sample data allows us to better comprehend the consequences of not specifying φ . In fact, we demonstrate that a suitable collection of XML sample documents reduces the decision problem of specifying φ to the problem of assessing the relevance of the sample documents for the application domain. In this paper, we address this reduction for Boolean constraints in XML. Our main contribution is the observation that propositional tableaux can be applied to generate all XML documents (up to equivalence) that form a counterexample for the implication of φ by Σ . If designers and users agree that a single such XML data tree is relevant for the underlying application domain, then φ can be discarded. In case all these generated counterexample documents are irrelevant, then φ should be specified explicitly.

1.1 Related Work

While Armstrong databases have been well-studied in the context of the relational data model (Fagin, 1982), such studies have not been conducted for XML constraints. Armstrong databases provide an excellent tool for verifying design decisions based on a user-friendly representation of constraint sets. In particular, they allow designers or users to discard candidate constraints. However, no ultimate conclusion can be drawn to specify the constraint. In this sense, Armstrong databases are not a good fit for the process of elimination outlined in Figure 1. Moreover, there is no guarantee at all that Armstrong databases exist, e.g., Boolean dependencies in relational databases do not enjoy Armstrong relations (Demetrovics *et al*, 1991a), and the problem of deciding whether an Armstrong relation exists for an arbitrary set of Boolean dependencies is NP-hard (Demetrovics *et al*, 1991b). Even for classes of constraints that do enjoy Armstrong relations, an

Armstrong database may contain exponentially many tuples in the number of attributes, e.g. for the class of minimal keys (Demetrovics, 1980; Katona and Tichler, 2006). Hence, it becomes difficult to focus on the candidate constraint under inspection.

Our approach is different in the sense that we only violate the candidate constraint in question (if possible) but not all the other constraints not implied by Σ . This enables the users and designers to focus on the relevance of the candidate constraint for the application domain. Furthermore, since our approach enables us to represent all possibly-relevant XML sample data (up to equivalence) we can recommend to specify the constraint.

In this article, we address constraint acquisition for Boolean constraints over XML documents (Hartmann *et al*, 2008). This class of constraints can be specified on arbitrary nodes of an XML schema graph, and can capture fundamental dependencies between different subnodes. Boolean constraints generalize Boolean dependencies that have been studied for relational databases (Demetrovics *et al*, 1991a; Sagiv *et al*, 1981) to the framework of XML. Boolean constraints cannot be specified by common XML schema languages such as DTDs (Bray *et al*, 2006) or XML Schema (Thomson *et al*, 2004), and cannot be modelled by any other XML constraints proposed in the literature, e.g., Arenas and Libkin (2004); Buneman *et al* (2001); Fan and Simeon (2003); Fan (2005); Hartmann and Trinh (2006); Hartmann and Link (2007); Vincent *et al* (2004). The implication problem of Boolean constraints is coNP-complete, which is an indication for the expressiveness of these constraints (Hartmann *et al*, 2008). In the present article, we explore their strong correspondence to propositional logic (Hartmann *et al*, 2008). Such logical characterisations have also been established for relational dependencies, e.g. Demetrovics *et al* (1991a); Sagiv *et al* (1981), and dependencies in complex-value databases, e.g. Hartmann and Link (2008).

Our techniques may become even more powerful when integrated into more general approaches. For instance, the work in Albrecht *et al* (1995) describes a general framework for constraint acquisition combining several techniques such as natural language processing, dialogues, heuristics and sample data. However, in their approach sample data is not generated automatically but entered by the designer. Moreover, tableaux techniques can help generating sufficient sample data such that candidate constraints cannot only be discarded, but also be specified.

1.2 Organisation

We will sketch an XML graph model in Section 2. The class of Boolean constraints is introduced in Section 3. We also include a brief summary of propositional logic, and outline the correspondence between the implication of Boolean constraints and the logical implication of propositional formulae. A summary of propositional tableaux techniques is given in Section 4. Finally, we outline how to assist Boolean constraint acquisition for XML in Section 5. We conclude in Section 6 and outline future work.

2. AN XML TREE MODEL

In this paper we follow a fairly simple XML graph model (Hartmann and Link, 2003) that is still powerful enough to capture important semantic information. We assume basic familiarity with notions from graph theory such as graphs, trees and paths (Jungnickel, 1999). Note that all graphs in this paper are considered to be finite, directed and without parallel arcs.

2.1 XML Graphs and Trees

For a graph G , let V_G denote its set of vertices and A_G its set of arcs. A *rooted graph* is a graph G with a distinguished vertex r_G , called the *root* of G , such that there is a directed path from r_G to every

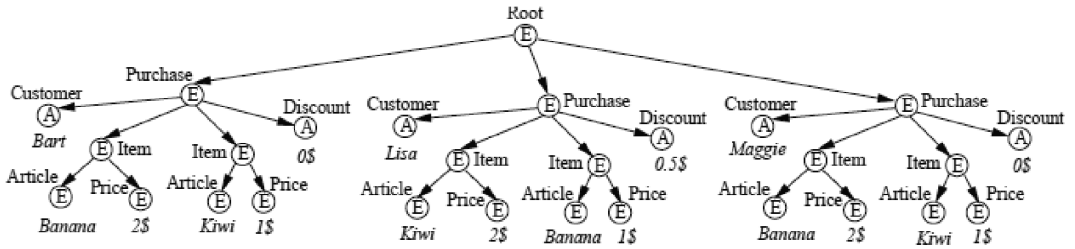


Figure 2: An XML data tree

other vertex in V_G . A *rooted tree* is a rooted graph T without any (non-directed) cycles. A graph G is *empty* if A_G is empty. Specifically, G is an *empty rooted graph* if it consists of a single vertex r_G . For every vertex v , let $Succ_G(v)$ denote its (possibly empty) set of successors, called *children*, in G . A non-isolated vertex without children is a *leaf* of G . Let L_G denote the set of all leaves of G .

Rooted graphs are frequently used to illustrate the structure of XML documents (Arenas and Libkin, 2004; Fan, 2005; Lee *et al*, 2002). In general, we assume that there are fixed sets $ENames$ of element names and $ANames$ of attribute names. An *XML graph* is a rooted graph G together with two mappings $name:V_G \rightarrow ENames \cup ANames$ and $kind:V_G \rightarrow \{E,A\}$ assigning every vertex its name and kind, respectively. If G is a rooted tree, we also speak of an *XML tree*. The symbols E and A tell us whether a vertex represents an element or attribute, respectively. We suppose that vertices of kind A are always leaves, while vertices of kind E can be leaves or non-leaves. In addition, we suppose that vertices of kind E have a name from $ENames$, while vertices of kind A have a name from $ANames$.

An *XML data tree* is an XML tree T together with an evaluation $val:L_T \rightarrow STRING$ assigning every leaf a (possibly empty) string, see Figure 2.

An *XML schema graph* is an XML graph G together with a mapping $freq:A_G \rightarrow \{?,1,*,+\}$ assigning every arc its frequency. By convention, arcs terminating in vertices of kind A should have frequency $?$ or 1 . In the left-most part of Figure 4, for example, we marked all arcs with their frequency, except those of frequency 1. Further, we suppose that no vertex in an XML schema graph G has two successors sharing both their kind and their name. Hence, the first graph in Figure 3 may serve as an XML schema graph, while the second one may not. If G is an XML tree we also speak of an *XML schema tree*.

2.2 Homomorphisms

Let G' and G be two XML graphs, and consider a mapping $\varphi:V_{G'} \rightarrow V_G$. We call the mapping φ *root-preserving* if the root of G' is mapped to the root of G , that is, $\varphi(r_{G'}) = r_G$. Further, φ is *kind-preserving* if the image of a vertex is of the same kind as the vertex itself, that is $kind(v') =$

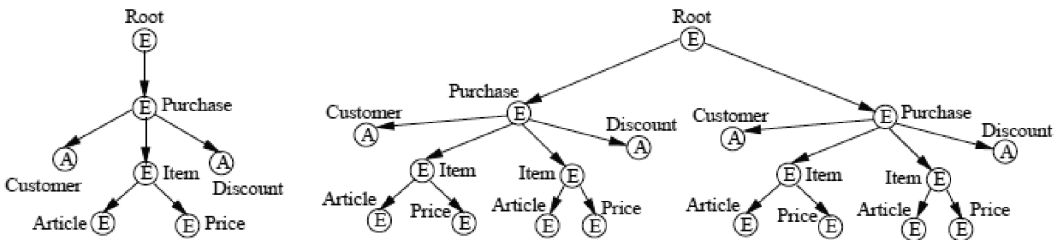


Figure 3: Two XML trees with vertex labels illustrating names and kinds of vertices.

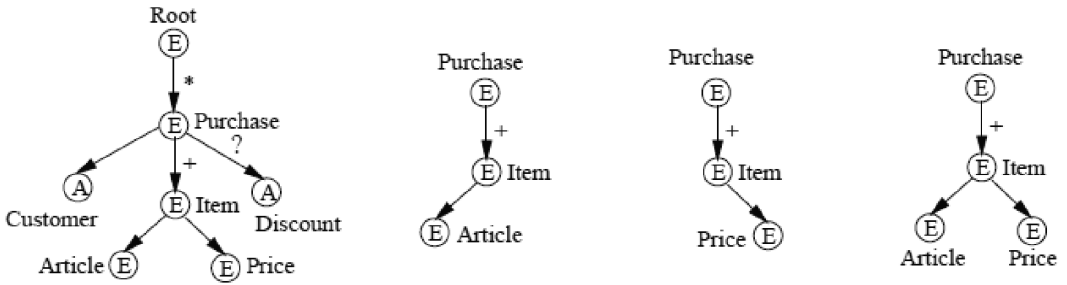


Figure 4: XML schema graph and three $v_{Purchase}$ -subgraphs $[[Article]]$, $[[Price]]$ and $[[Article, Price]]$

$kind(\varphi(v'))$ for all $v' \in V_{G'}$. Finally, φ is *name-preserving* if the image of a vertex carries the same name as the vertex itself, that is, $name(v') = name(\varphi(v'))$ for all $v' \in V_{G'}$. The mapping φ is a *homomorphism* between G' and G if the following conditions hold:

- (1) every arc of G' is mapped to an arc of G , that is $(v', w') \in A_{G'}$ implies $(\varphi(v'), \varphi(w')) \in A_G$,
- (2) φ is root-, kind- and name-preserving.

A homomorphism φ may be naturally extended to the arc set of G' : given an arc $a' = (v', w')$ of G' , let $\varphi(a')$ denote the arc $(\varphi(v'), \varphi(w'))$ of G .

Example 2.1: Consider the two XML trees in Figure 3. There is a unique name-preserving mapping φ which maps the vertices of the second graph G' to the vertices of the first graph G . That is, all vertices with name *Purchase* in the second graph are mapped to the single vertex with name *Purchase* in the first graph, etc. It is not difficult to verify that this mapping satisfies conditions (1) and (2) above, i.e., φ is indeed a homomorphism.

Let T' be an XML data tree and G an XML schema graph. T' is said to be *compatible* with G if there is a homomorphism $\phi: V_{T'} \rightarrow V_G$ between T' and G such that for each vertex v' of T' and each arc $a = (\varphi(v'), w)$ of G , the number of arcs $a' = (v', w_i)$ mapped to a is at most 1 if $freq(a) = ?$, exactly 1 if $freq(a) = 1$, at least 1 if $freq(a) = +$, and arbitrarily many if $freq(a) = *$. Due to the definition of XML schema graph, this homomorphism is unique if it exists.

Example 2.2: For example, consider the XML data tree T' from Figure 2 and the XML schema tree T in Figure 4. Again, the unique name-preserving mapping from $V_{T'}$ to V_T is a homomorphism between T' and T . On comparing both trees and checking the frequencies, it turns out that T' is compatible with T .

A homomorphism $\varphi: V_{G'} \rightarrow V_G$ between XML graphs G' and G is an *isomorphism* if φ is bijective and φ^{-1} is a homomorphism, too. Then G' is said to be *isomorphic* to G or a *copy* of G . We call two isomorphic XML data trees T' and T *value-equal* if the isomorphism $\varphi: V_{T'} \rightarrow V_T$ between them is also *evaluation-preserving*, i.e., $val(v') = val(\varphi(v'))$ holds for every vertex $v' \in V_{T'}$.

2.3 v -subgraphs

Let G be a rooted graph, v be a vertex of G , and $L_U \subseteq L_G$ be a set of leaves of G . Consider the union U of all directed paths of G from v to some leaf in L . We call U a *v -subgraph* of G . Clearly, every non-empty v -subgraph of an XML graph is itself an XML graph. In particular, a non-empty v -subgraph of an XML tree is again an XML tree. We use $Sub_G(v)$ to denote the set of all v -subgraphs of G . If v is clear from the context, then we denote the v -subgraph U as $[[l_1, \dots, l_k]]$ where $\{l_1, \dots, l_k\} = L_U$.

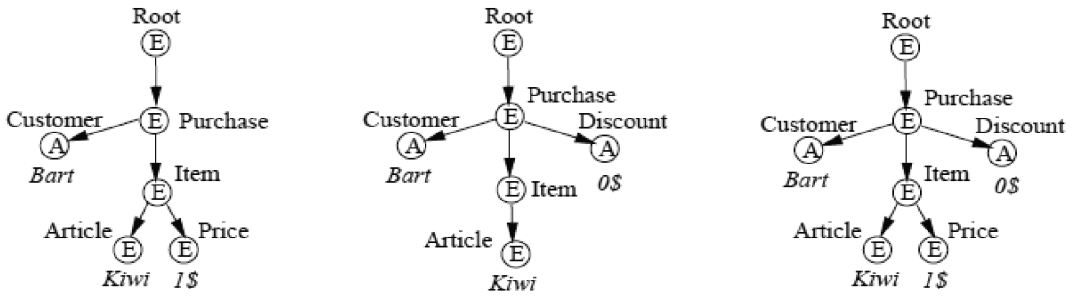


Figure 5: Three subcopies of the XML schema graph in Figure 4.

In particular, $[[\emptyset]]$ denotes the empty v -subgraph. For example, in Figure 4 we have the three $v_{Purchase}$ -subgraphs $[[Article]]$, $[[Price]]$ and their v -subgraph union $[[Article]] \cup [[Price]] = [[Article, Price]]$.

If U contains all leaves of G which may be reached from v by passing a directed path of G , then U is said to be the *total v -subgraph* of G and is denoted by $G(v)$. Note that $G(v)$ is maximal among the v -subgraphs of G and contains every other v -subgraph of G as a v -subgraph itself.

Let G and G' be XML graphs. A homomorphism $\varphi: V_{G'} \rightarrow V_G$ between them induces a mapping of the total subgraphs of G' to the total subgraphs of G : given a total v' -subgraph $G'(v')$ of G' , $\varphi(G'(v'))$ denotes the total $\varphi(v')$ -subgraph $G(\varphi(v'))$ of G . Note that the vertices and arcs of $G'(v')$ are mapped to the vertices and arcs of $\varphi(G'(v'))$. Obviously, the number of pre-images of a total v -subgraph of G coincides with the number of pre-images of the vertex v .

Let G' and G be two XML graphs together with a homomorphism φ between them. An $r_{G'}$ -subgraph U' of G' is a *subcopy* of G if U' is isomorphic to some r_G -subgraph U of G , and an *almost-copy* of G if it is maximal with this property. Almost-copies are of special interest as an XML data tree compatible with some XML schema tree T does not necessarily contain copies of T .

Example 2.3: Let T denote the XML schema graph of Figure 4, and T' denote the T -compatible XML data tree from Figure 2. Each of the three XML data trees in Figure 5 is a subcopy of T . Comparing all three of them only the subcopy on the very right is an almost-copy of T (the graphs on the left and in the middle are subgraphs of the graph on the right). In fact, the almost-copy is a copy of T .

2.4 Projections

Let G' and G be two XML graphs together with a homomorphism $\varphi: V_{G'} \rightarrow V_G$ between them. Given an r_G -subgraph U of G , the *projection* of G' to the subgraph U is the union of all subcopies of U in

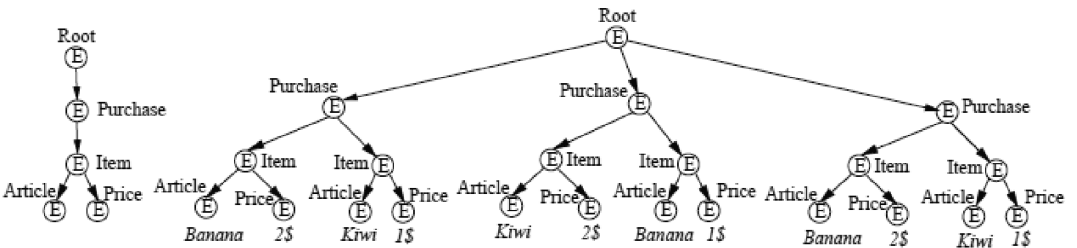


Figure 6: Projection of the data tree in Figure 2 to the subgraph on its left.

G' , and denoted by $G'|_U$. In particular, $G'|_U$ is an $r_{G'}$ -subgraph of G' .

For instance, let G' denote the XML data tree in Figure 2. G' is compatible with the XML schema graph G in Figure 4. Let further U denote the r_G -subgraph of G that is illustrated in the left of Figure 6. Then the projection $G'|_U$ is shown in the right of Figure 6.

3. XML CONSTRAINTS

We will review the syntax and semantics of Boolean constraints in XML (Hartmann and Link, 2008). Therefore, it is necessary to understand which ν -subgraphs of an XML data tree suffice to identify pre-images of the total ν -subgraph up to equivalence. We will review this problem first.

3.1 Equivalence of XML Data

Let T be some XML schema tree, ν be a vertex of T and T' be some XML data tree compatible with T . In this section, we will answer the question which ν -subgraphs of T suffice to identify pre-images of $T(\nu)$ in T' up to equivalence. More precisely, what is the minimal set $\mathcal{E}(\nu) \subseteq \text{Sub}_T(\nu)$ such that equivalence between two arbitrary pre-images W_1, W_2 of $T(\nu)$ in T' can be determined by the equivalences of $W_1|_X$ and $W_2|_X$ on all $X \in \mathcal{E}(\nu)$? In other words, for which minimal $\mathcal{E}(\nu)$ is it true that if W_1 and W_2 are not equivalent, then there is some $X \in \mathcal{E}(\nu)$ such that $W_1|_X$ and $W_2|_X$ are already not equivalent?

The set $\mathcal{E}(\nu)$ should not consist of all ν -subgraphs of T since projections on some ν -subgraphs uniquely determine projections on other ν -subgraphs. This is illustrated by the following example.

Example 3.1: Consider the XML schema tree in Figure 4. The projections of any ν_{Purchase} -subgraphs on $\llbracket \text{Customer} \rrbracket$ and on $\llbracket \text{Article} \rrbracket$ suffice to decide the equivalence of the projections on the union $\llbracket \text{Customer}, \text{Article} \rrbracket$ of $\llbracket \text{Customer} \rrbracket$ and $\llbracket \text{Article} \rrbracket$. The reason for this is that the ν_{Purchase} -subgraphs on $\llbracket \text{Customer} \rrbracket$ and $\llbracket \text{Article} \rrbracket$ do not share any arc of multiple frequency. Consequently, the ν_{Purchase} -subgraph $\llbracket \text{Customer}, \text{Article} \rrbracket$ cannot belong to $\mathcal{E}(\nu)$.

In order to answer the initial question we seek some intuition from relational databases. Which subschemata of a relation schema allow us to uniquely identify tuples in a relational database? Suppose we have a relation schema $R = \{A_1, \dots, A_k\}$ with attributes A_1, \dots, A_k . Then every R -tuple $t: R \rightarrow \bigcup_{i=1}^k \text{dom}(A_i)$ with $t(A_i) \in \text{dom}(A_i)$ is uniquely determined by its projections $t(A_1), \dots, t(A_k)$. That is, there cannot be two different tuples which have the same projection on all the attributes. Consequently, the unary subschemata (those consisting of a single attribute) allow us to identify tuples.

With this in mind it is natural to conjecture that $\mathcal{E}(\nu) = \mathcal{U}(\nu)$, i.e., $\mathcal{E}(\nu)$ consists of exactly all unary ν -subgraphs of T , that is, all the directed paths from ν to a leaf of T . However, it turns out that this set is too small. We will demonstrate this by the following example.

Example 3.2: Figure 7 shows the three pre-images of the total ν_{Purchase} -subgraph of the XML schema tree from Figure 4 in the XML data tree T' from Figure 2. Two unary ν_{Purchase} -subgraphs are $\llbracket \text{Article} \rrbracket$ and $\llbracket \text{Price} \rrbracket$. All three pre-images have the same projection on $\llbracket \text{Article} \rrbracket$ and on $\llbracket \text{Price} \rrbracket$. However, the second and third pre-image still deviate in their projection on $\llbracket \text{Article}, \text{Price} \rrbracket$. In this sense, the projection on $\llbracket \text{Article} \rrbracket$ and the projection on $\llbracket \text{Price} \rrbracket$ do not allow us to distinguish between the second and the third ν_{Purchase} pre-image. The reason for this is that $X = \llbracket \text{Article} \rrbracket$ and $Y = \llbracket \text{Price} \rrbracket$ do share an arc of multiple frequency.

We will now generalise the observation of **Example 3.2**.

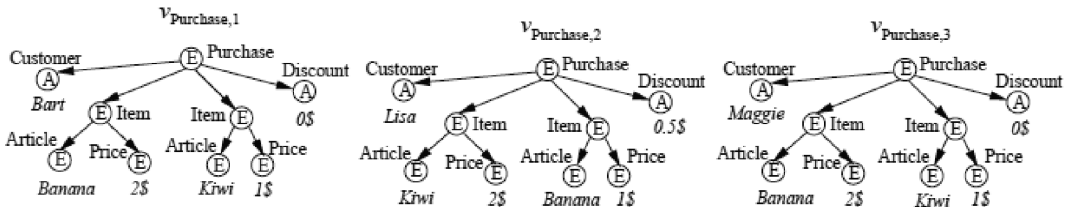


Figure 7: The three pre-images of the total $v_{Purchase}$ -subgraphs of the XML schema tree from Figure 4 in the XML data tree T' from Figure 2. All three pre-images have the same projection on $[[Article]]$ and on $[[Price]]$, but only the first and third pre-image have the same projection on $[[Article, Price]]$.

Definition 3.1: Let T be some XML schema tree, v be a vertex of T and $X, Y \in Sub_T(v)$. The v -subgraphs X and Y are called reconcilable if and only if there are v -subgraphs X' of X and Y' of Y such that X' and Y' share no arc of multiple frequency and such that $X' \sqcup Y' = X \sqcup Y$ holds.

This means, that whenever X and Y share some arc (u, w) of frequency $*$ or $+$, then X contains the total w -subtree of Y or Y contains the total w -subtree of X .

Example 3.3: The $v_{Purchase}$ -subgraphs $X = [[Article]]$ and $Y = [[Price]]$ both contain the arc $(v_{Purchase}, v_{Item})$ while their v_{Item} -subgraphs are obviously no subgraphs of one another, see Figure 4. Hence, $X = [[Article]]$ and $Y = [[Price]]$ are not reconcilable. On the other hand, however, $X = [[Customer]]$ and $Y = [[Article]]$ are reconcilable.

The next theorem justifies the syntactic definition of reconcilability in semantical terms. In fact, two v -subgraphs X and Y are reconcilable precisely if the projections on X and on Y uniquely determine the projection on $X \sqcup Y$.

Theorem 3.1: (Hartman et al, 2008) Let T be some XML schema tree, and v be a vertex of T . For all $X, Y \in Sub_T(v)$ we have that X and Y are reconcilable if and only if for all XML data trees T' that are compatible with T and for all pre-images W, W' of $T(v)$ in T' the following holds: if $W|_X$ is equivalent to $W'|_X$ and $W|_Y$ is equivalent to $W'|_Y$, then $W|_{X \sqcup Y}$ is equivalent to $W'|_{X \sqcup Y}$.

The following example shows an instance of **Theorem 3.1** where X and Y are not reconcilable.

Example 3.4: Consider the pre-images W rooted at $v_{Purchase,1}$ and W' rooted at $v_{Purchase,2}$ in Figure 7. Their projections on $X = [[Article]]$ and on $Y = [[Price]]$ are equivalent, but their projections on $X \sqcup Y = [[Article, Price]]$ are not equivalent.

It now remains to define the set of v -subgraphs that are necessary and sufficient to decide equivalence between arbitrary pre-images.

Definition 3.2: Let T be some XML schema tree, and v be a vertex of T . The set $\mathcal{E}(v) \subseteq Sub_T(v)$ of essential subgraphs of v is defined as the smallest set of v -subgraphs of T with the following two properties: (i) $\mathcal{U}(v) \subseteq \mathcal{E}(v)$, and (ii) if $X, Y \in \mathcal{E}(v)$ are not reconcilable, then $X \sqcup Y \in \mathcal{E}(v)$.

The essential subgraphs of v form therefore the smallest set that contains the unary subgraphs of v and that is closed under the union of v -subgraphs that are not reconcilable. This seems now very natural: for any $X, Y \in \mathcal{E}(v)$ for which the two projections $W|_X$ and $W|_Y$ of some pre-image W of v

do not uniquely determine the projection of $W|_{X \sqcup Y}$, the subgraphs X and Y cannot be reconcilable, and $X \sqcup Y$ is therefore included in $\mathcal{E}(v)$.

Example 3.5: Let T denote the XML schema tree in Figure 4. Consequently, $\mathcal{E}(v_{Purchase})$ consists of the following $v_{Purchase}$ -subgraphs:

[[Customer]], [[Article]], [[Price]], [[Discount]], and [[Article,Price]].

The binary $v_{Purchase}$ -subgraph [[Article,Price]] belongs to $\mathcal{E}(v_{Purchase})$ since [[Article]] and [[Price]] are not reconcilable. It follows that, in order to determine any pre-image of $T(v_{Purchase})$ up to equivalence, only up to five different projections are necessary.

The analysis in this section will give us invaluable information for defining Boolean constraints in the next section.

3.2 Boolean Constraints

The following definition establishes the syntactic structure of Boolean constraint expressions. The results of the last section suggest to utilize essential v -subgraphs as the atoms of a Boolean constraint over v .

Definition 3.3: Let T be an XML schema tree, and v be a vertex of T . The set of Boolean constraints over the vertex v is defined as the smallest set $BC(v)$ with the following properties:

- if $X \in \mathcal{E}(v)$ then $v:X \in BC(v)$,
- if $\varphi \in BC(v)$ then $v:\neg\varphi \in BC(v)$, and
- if $v:\varphi, v:\psi \in BC(v)$ then $v:(\varphi \wedge \psi) \in BC(v)$.

For the sake of readability we introduce the following abbreviations: $v:(\varphi \vee \psi)$ stands for $v:\neg(\neg\varphi \wedge \neg\psi)$ and $v:(\varphi \Rightarrow \psi)$ stands for $v:(\neg\varphi \vee \psi)$.

Example 3.6: Consider the XML schema tree in Figure 4. Some Boolean constraints over $v_{Purchase}$ are (we omit outmost parentheses):

- $\neg[[Customer]]$,
- $[[Article,Price]] \Rightarrow [[Discount]]$, and
- $[[Discount]] \Rightarrow [[Customer]] \vee [[Article,Price]]$.

The expression $v_{Purchase}: [[Customer,Article,Price]] \Rightarrow [[Discount]]$ is not a Boolean constraint over $v_{Purchase}$ because $[[Customer,Article,Price]]$ is not an essential subgraph of $v_{Purchase}$.

The next definition will assign a semantics to Boolean constraints. We will define when a compatible XML tree satisfies a Boolean constraint.

Definition 3.4: Let T be an XML schema tree, v be a vertex of T , and T' be an XML data tree that is compatible with T . Two distinct pre-images W_1, W_2 of $T(v)$ in T' are said to satisfy the Boolean constraint φ over v , denoted by $\mathbb{F}_{\{W_1, W_2\}} \varphi$, if and only if the following holds:

- if $\varphi = v:X$ for some $X \in \mathcal{E}(v)$, then $\mathbb{F}_{\{W_1, W_2\}} \varphi$ if and only if $W_1|_X$ and $W_2|_X$ are equivalent,
- if $\varphi = v:\neg\psi$ for some $\psi \in BC(v)$, then $\mathbb{F}_{\{W_1, W_2\}} \varphi$ if and only if not $\mathbb{F}_{\{W_1, W_2\}} v:\psi$,
- if $\varphi = v:(\psi_1 \wedge \psi_2)$ for $\psi_1, \psi_2 \in BC(v)$ then $\mathbb{F}_{\{W_1, W_2\}} \varphi$ if and only if $\mathbb{F}_{\{W_1, W_2\}} v:\psi_1$ and $\mathbb{F}_{\{W_1, W_2\}} v:\psi_2$.

We say that T' satisfies the Boolean constraint φ over v , denoted by $\vDash_{T'} \varphi$, if and only if for all distinct pre-images W_1, W_2 of $T(v)$ in T' we have that $\vDash_{\{W_1, W_2\}} \varphi$.

The following example illustrates **Definition 3.4** and exemplifies the reason for using essential subgraphs as atoms of Boolean constraints.

Example 3.7: Recall from **Example 3.6** that the expression

$$v_{Purchase}: [[Customer, Article, Price]] \Rightarrow [[Discount]]$$

is not a Boolean constraint over $v_{Purchase}$. However, as we can see from **Definition 3.4** this does not result in a loss of expressiveness. In fact, a tree compatible with the schema tree in Figure 4 satisfies $v_{Purchase}: [[Customer, Article, Price]] \Rightarrow [[Discount]]$ (in the obvious sense) if and only if the tree satisfies the Boolean constraint $v_{Purchase}: [[Customer]] \wedge [[Article, Price]] \Rightarrow [[Discount]]$.

It is the goal of this paper to provide decision support for specifying Boolean constraints, i.e., given some set $\Sigma \subseteq BC(v)$ of already specified constraints we want to provide support for deciding whether a candidate constraint $\varphi \in BC(v)$ captures relevant information on the application domain that is not already captured implicitly by Σ .

Therefore, we will use a correspondence between the implication of Boolean constraints and the logical implication of propositional formulae (Hartmann *et al*, 2008). Hence, the remainder of this section is devoted to describing this correspondence since it is fundamental to our method.

The implication of Boolean constraints is defined as followed. Let T be an XML schema tree, v be a vertex of T , and $\Sigma \cup \{\varphi\} \subseteq BC(v)$. We say that Σ implies φ if every T -compatible XML data tree T' that satisfies all constraints in Σ also satisfies φ . Besides illustrating the definition of implication for Boolean constraints the next example also shows why it can be quite difficult to specify the correct Boolean constraints, i.e., the constraints that are relevant for the application domain.

Example 3.8: The Boolean constraint

$$v_{Purchase}: [[Discount]] \Rightarrow ([[Customer]] \vee [[Article, Price]])$$

implies the Boolean constraint

$$v_{Purchase}: (\neg [[Customer]] \wedge \neg [[Article, Price]]) \Rightarrow \neg [[Discount]]$$

and vice versa. However, the Boolean constraint

$$v_{Purchase}: ([[Article]] \wedge [[Price]]) \Rightarrow \neg [[Discount]]$$

is not implied by the Boolean constraint

$$v_{Purchase}: ([[Article, Price]]) \Rightarrow \neg [[Discount]]$$

since there is some XML tree that satisfies the latter but violates the first constraint.

3.3 Propositional Logic

We will now fix the basic notions and notations from classical Boolean propositional logic (Enderton, 2001). Let \mathcal{V} denote a set of propositional variables. The set $\mathbb{F}_{\mathcal{V}}$ of propositional formulae over \mathcal{V} is the smallest set with the following properties:

- every propositional variable in \mathcal{V} is a formulae in $\mathbb{F}_{\mathcal{V}}$,
- if $\varphi \in \mathbb{F}_{\mathcal{V}}$, then $\neg \varphi \in \mathbb{F}_{\mathcal{V}}$,
- if $\psi'_1, \psi'_2 \in \mathbb{F}_{\mathcal{V}}$ then $(\psi'_1 \wedge \psi'_2) \in \mathbb{F}_{\mathcal{V}}$.

We will use 0, 1 and *false*, *true* interchangeably to denote propositional truth values. A truth assignment over \mathcal{V} is a mapping $\theta: \mathcal{V} \rightarrow \{0,1\}$ that assigns each variable in \mathcal{V} a truth value. A truth assignment over \mathcal{V} is extended to a function $\Theta: \mathcal{V} \rightarrow \{0,1\}$ that maps every formula φ' in $\mathbb{F}_{\mathcal{V}}$ to its truth value $\Theta(\varphi')$ as follows:

- if $\varphi' \in \mathcal{V}$, then $\Theta(\varphi') = \theta(\varphi')$,
- if $\varphi' = \neg\psi$ for $\psi \in \mathbb{F}_{\mathcal{V}}$, then $\Theta(\varphi') = \begin{cases} 1, & \text{if } \Theta(\psi) = 0 \\ 0, & \text{otherwise} \end{cases}$,
- if $\varphi' = \psi'_1 \wedge \psi'_2$ for $\psi'_1, \psi'_2 \in \mathbb{F}_{\mathcal{V}}$ then $\Theta(\varphi') = \begin{cases} 1, & \text{if } \Theta(\psi_1) = \Theta(\psi_2) = 1 \\ 0, & \text{otherwise} \end{cases}$.

Note that we will make frequent use of the abbreviations $(\varphi' \vee \psi')$ for $\neg(\neg\varphi' \wedge \neg\psi')$ and $(\varphi' \Rightarrow \psi')$ for $(\neg\varphi' \vee \psi')$, similar to the case of Boolean constraints.

We say that a truth assignment θ over \mathcal{V} *satisfies* the formulae φ' in $\mathbb{F}_{\mathcal{V}}$, denoted by $\models_{\theta} \varphi'$, if and only if $\Theta(\varphi') = 1$. If $\models_{\theta} \varphi'$, then we call θ a *model* of φ' . If θ does not satisfy φ we sometimes say that θ *violates* φ . We say that θ *satisfies* the set Σ' of formulae in $\mathbb{F}_{\mathcal{V}}$ (or that θ is a model of Σ') if and only if θ satisfies every φ' in Σ' .

We further say that a set Σ' of propositional formulae over \mathcal{V} *logically implies* the propositional formula φ' over \mathcal{V} if and only if every model of Σ' is also a model of φ' . In other words, there is no truth assignment that satisfies all the formulae in Σ' and does not satisfy φ' .

3.4 The Correspondence

Let $\gamma: \mathcal{E}(v) \rightarrow \mathcal{V}$ denote a bijection between the essential subgraphs of v and the set \mathcal{V} of propositional variables. We will now extend this bijection to Boolean constraints over v and the set $\mathbb{F}_{\mathcal{V}}$ of propositional formulae over \mathcal{V} . We recursively define this mapping $\Gamma: BC(v) \rightarrow \mathbb{F}_{\mathcal{V}}$ for Boolean constraints φ to their propositional formulae $\Gamma(\varphi) = \varphi'$. If $\varphi = v: X$ for some $X \in \mathcal{E}(v)$, then let $\varphi' = \gamma(X)$. The rest of the mapping is straightforward: for $\varphi = v: \neg\psi$ we have $\varphi' = v: \neg\psi'$, and for $\varphi = v: (\psi_1 \wedge \psi_2)$ we have $\varphi' = v: (\psi'_1 \wedge \psi'_2)$. If Σ is a set of Boolean constraints over v , then let $\Sigma' = \{\sigma' \mid \sigma \in \Sigma\}$ denote the corresponding set of propositional formulae over \mathcal{V} . Furthermore, the set $\Sigma'_v = \{\gamma(X) \Rightarrow \gamma(Y) \mid X, Y \in \mathcal{E}(v), X \text{ covers } Y\}$ denotes those formulae which encode the structure of (the essential subgraphs of) v . Note that X covers Y iff Y is a v -subgraph of X and for all $Z \in \mathcal{E}(v)$ where Y is a v -subgraph of Z and Z is a v -subgraph of X we have $Y = Z$ or $X = Z$.

Example 3.9: Consider the XML schema tree T in Figure 4, vertex $v_{Purchase}$,

$$\Sigma = \{v_{Purchase}: [[\text{Article}, \text{Price}]] \Rightarrow [[\text{Discount}]]\}$$

and $\varphi = v_{Purchase}: [[\text{Article}]] \wedge [[\text{Price}]] \Rightarrow [[\text{Discount}]]$. The T -compatible XML data tree T' on the bottom of Figure 9 shows that φ is not implied by Σ . We define the mapping $\gamma: \mathcal{E}(v_{Purchase}) \rightarrow \mathcal{V}$ as follows:

- $\gamma([[Customer]]) = V_1$,
- $\gamma([[Article]]) = V_2$,
- $\gamma([[Price]]) = V_3$,
- $\gamma([[Discount]]) = V_4$, and
- $\gamma([[Article, Price]]) = V_5$.

The resulting mappings into propositional formulae are then as follows:

- $\Sigma' = \{V_5 \Rightarrow V_4\}$,
- $\Sigma'_{v_{Purchase}} = \{V_5 \Rightarrow V_2, V_5 \Rightarrow V_3\}$, and
- $\varphi' = (V_2 \wedge V_3) \Rightarrow V_4$.

We can notice that φ' is not implied by Σ' and $\Sigma'_{v_{Purchase}}$. In fact, the truth assignment $\theta_{T'}$, with $\theta_{T'}(V_i) = \text{true}$ if and only if $i \in \{1,2,3\}$ satisfies all the formulae in $\Sigma' \cup \Sigma'_{v_{Purchase}}$, but violates φ' .

The point here is that the two pre-images of $T(v_{Purchase})$ in T' have isomorphic projections on precisely those essential $v_{Purchase}$ -subgraphs of T whose corresponding variables are interpreted as true by $\theta_{T'}$.

The last example illustrates the correspondence between XML trees that form a counterexample for the implication of a Boolean constraint φ by a set Σ of Boolean constraints and truth assignments that form a counterexample for the logical implication of the propositional formula φ' by the set Σ' of propositional formulae. This correspondence holds in general.

Theorem 3.2 (Hartmann et al, 2008): *Let be an XML schema tree, v be a vertex of T , and $\Sigma \cup \{\varphi\} \subseteq BC(v)$ be a set of Boolean constraints over v . Let Σ'_v denote the propositional formulae which encode the structure of v , and Σ' denote the corresponding set of propositional formulae for Σ . Then Σ implies φ if and only if $\Sigma' \cup \Sigma'_v$ logically implies φ' .*

Let T be an XML schema tree, v be a vertex of T and T' be a T -compatible XML data tree. A two pre-image projection of T' with respect to v is an $r_{T'}$ -subgraph of T' that is the union of i) two distinct pre-images of $T(v)$ in T' and ii) the union of the paths from the root $r_{T'}$ of T' to the roots of the two pre-images. A T -compatible XML data tree T' is said to be a two pre-image data tree with respect to v if and only if there are two distinct pre-images of $T(v)$ in T' and T' is isomorphic to each of its two pre-image projections with respect to v .

The following lemma reduces the implication problem of Boolean constraints over v from all compatible XML data trees T' to two pre-image data trees. It is important to our approach since it i) provides us with optimal sample documents (those that contain only two pre-images) and ii) shows the sufficiency of analysing all two pre-image data trees.

Lemma 3.1 (Hartmann et al, 2008): *Let T be an XML schema tree, v be a vertex of T , and $\Sigma \cup \{\varphi\} \subseteq BC(v)$ be a set of Boolean constraints over v . Suppose that T' is a T -compatible XML data tree that satisfies all constraints in Σ but violates φ . Then there is a two pre-image projection \bar{T} of T' with respect to v such that i) \bar{T} satisfies all constraints in Σ , and ii) \bar{T} violates φ .*

Figure 8 shows an example of a two pre-image projection of the XML data tree T' of Figure 2. Indeed, T' is the union of the two pre-images rooted at $v_{Purchase,1}$ and $v_{Purchase,3}$, respectively, in Figure 7, and the union of the paths from $r_{T'}$ to $v_{Purchase,1}$ and $v_{Purchase,3}$, respectively.

4. PROPOSITIONAL TABLEAUX

We will summarise the extremely elegant and efficient proof procedure for propositional logic that is known as *analytical tableaux*. The technique goes back to Beth (1959), Hintikka (1955) and its idea derives from Gentzen (1935). For a comprehensive view on propositional and first-order logic from an analytical tableaux point of view we recommend Smullyan (1995).

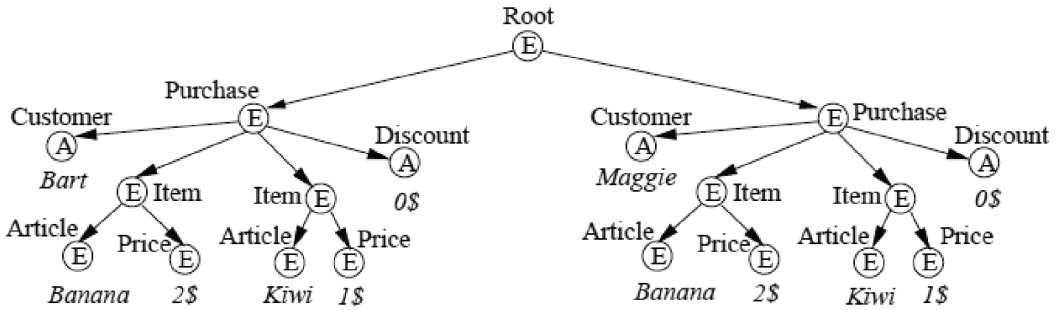
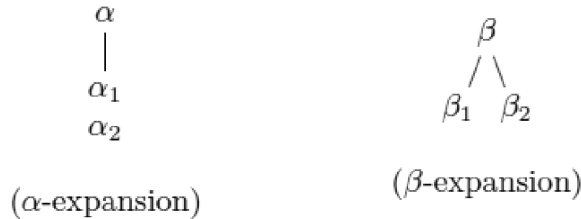


Figure 8: A two pre-image projection of the data tree in Figure 2 with respect to the $v_{purchase}$ -vertex in the XML schema tree of Figure 4.

We now summarise the rules that apply to the formulae in a tableau and expand the tableau at a leaf. The conclusions of a rule are appended either vertically or horizontally at a leaf whenever the premise of the expansion rule matches a formula that appears anywhere on the path from the root to that leaf. We distinguish between an α -expansion and a β -expansion:



In an α -expansion both conclusions α_1 and α_2 are appended on top of one another, whereas in a β -expansion the tree branches into β_1 and β_2 . Table 1 shows the different formulae for an α -expansion.

α	α_1	α_2
$X \wedge Y$	X	Y
$\neg(X \vee Y)$	$\neg X$	$\neg Y$
$\neg(X \Rightarrow Y)$	X	$\neg Y$
$\neg(\neg X)$	X	X

Table 1: α -expansion

We note that under any truth assignment, α is *true* precisely if α_1 and α_2 are both *true*. Table 2 shows the different formulae for a β -expansion.

β	β_1	β_2
$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$X \vee Y$	X	Y
$X \Rightarrow Y$	$\neg X$	Y

Table 2: β -expansion

Under any truth assignment, β is *true* precisely if at least one of the pair β_1 and β_2 is *true*.

An *analytical tableau* is a marked (by propositional formulae), unordered, finite tree that is inductively defined as follows:

- The tree consisting of a single path is a tableau for $\{F_1, F_2, \dots, F_n\}$:

$$\begin{array}{c} F_1 \\ F_2 \\ \vdots \\ F_n \end{array}$$

- If T is a tableau for $\{F_1, F_2, \dots, F_n\}$, and T' results from T by applying an expansion rule then T' is also a tableau for $\{F_1, F_2, \dots, F_n\}$.

A *branch* of a tableau is a maximal path, i.e., a path from the root to a leaf. A branch is said to be *closed* if it contains a variable and its negation. Closed branches are marked by a cross \times at the bottom of the branch. A tableau is called *closed* precisely when all its branches are closed. A branch ϑ of a tableau is *complete* if for every α which occurs in ϑ , both α_1 and α_2 occur in ϑ , and for every β which occurs in ϑ , at least one of β_1 and β_2 occurs in ϑ . We call a tableau *completed* if every branch is either closed or completed.

Theorem 4.1: *Every complete open branch of a tableau for $\Sigma' \cup \{\neg\varphi'\}$ defines a truth assignment θ that satisfies Σ' and violates φ' .*

We will now summarise the constructive proof of **Theorem 4.1** (for a full proof see for instance Smullyan (1995)) which has interesting consequences for us. Let S be the set of formulae in a complete open branch ϑ of a tableau. Then S satisfies the following three conditions for every α and β :

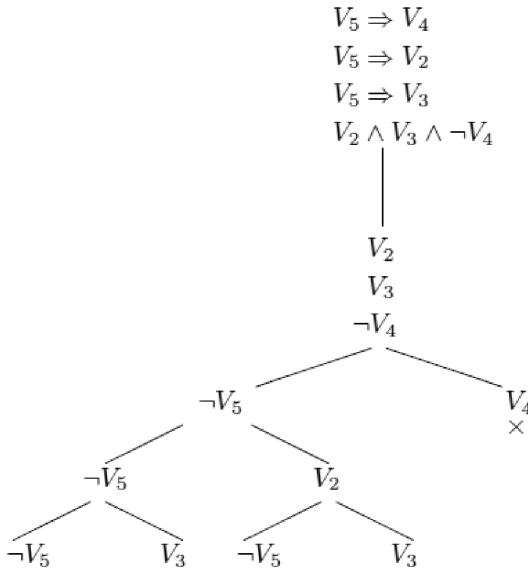
- No variable and its negation are both in S .
- If $\alpha \in S$, then $\alpha_1 \in S$ and $\alpha_2 \in S$.
- If $\beta \in S$, then $\beta_1 \in S$ or $\beta_2 \in S$.

Sets S obeying these three conditions are known as *Hintikka* sets which are satisfiable. In fact, we can define a truth assignment that makes every element of S true. Therefore, we assign each variable p , which occurs in at least one element of S , a truth value as follows:

- If $p \in S$, then give p the value *true*.
- If $\neg p \in S$, then give p the value *false*.
- If neither p nor $\neg p$ is an element in S , then give p the value *true* or *false* at will.

It can be shown that every element of S is true under this truth assignment.

Example 4.1: *We continue Example 3.9 by applying the method of analytical tableau to decide the implication of φ' by Σ' . Hence, we want to refute Σ' and $\neg\varphi'$. Then we obtain:*



In this tableau only the marked branch is closed, and all other branches define models of $\Sigma' \cup \{\neg\varphi'\}$.

Completed tableaux do not only allow us to read off a single truth assignment that satisfies Σ' and violates φ' , but *all* models of $\Sigma' \cup \{\neg\varphi'\}$. This observation is crucial for our semantic acquisition process.

Example 4.2: Consider the complete tableau from **Example 4.1**. The following truth assignments can be read off the tableau.

V_1	V_2	V_3	V_4	V_5
<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>false</i>
<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>false</i>

These form the set of all truth assignments that satisfy $V_5 \Rightarrow V_3$, $V_5 \Rightarrow V_2$, $V_5 \Rightarrow V_4$, but violate $(V_2 \wedge V_3) \Rightarrow V_4$.

5. DECISION SUPPORT FOR XML CONSTRAINT ACQUISITION

In this section, we will describe how off-the-shelf tools for solving problems in propositional logic can be utilised in the constraint acquisition process. First, we present the general approach. Subsequently, we will briefly comment on techniques that provide alternatives to propositional tableaux. Finally, we will make some remarks on the time complexity of our approach.

5.1 Assisting the Acquisition Process

Suppose that we would like to determine those Boolean constraints that need to be specified on the vertex v of our XML schema tree T . So far, we have gathered a certain set $\Sigma \subseteq BC(v)$ of Boolean constraints that should be specified (initially $\Sigma = \emptyset$). Some user or designer may suggest that another constraint φ captures important semantic information about the application domain. That is,

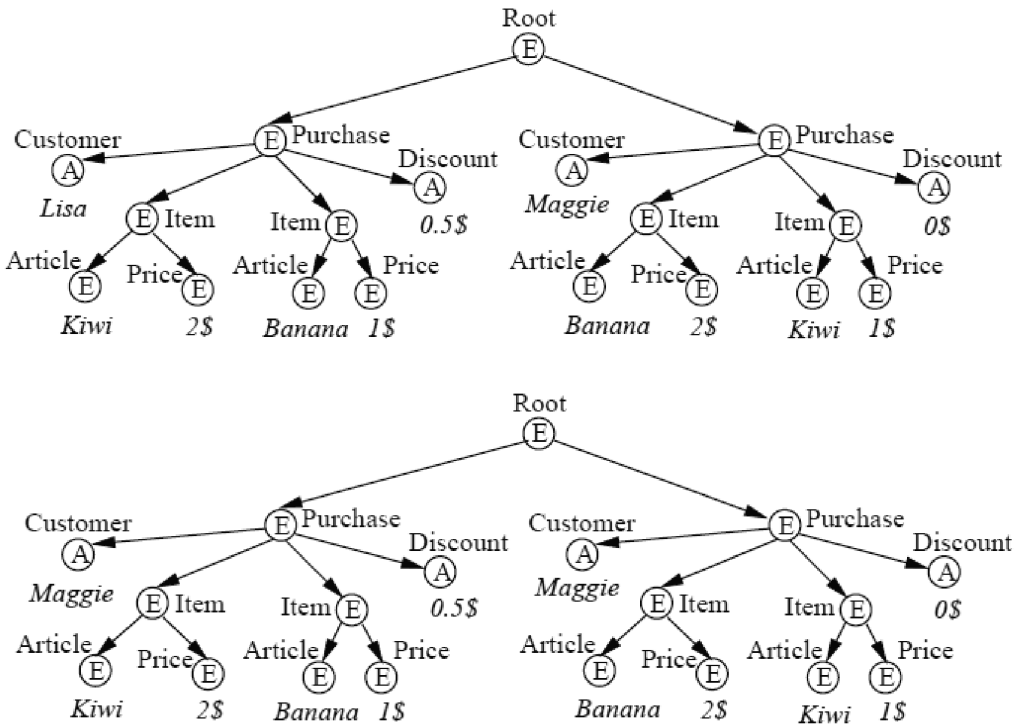


Figure 9: Two XML data trees

φ becomes the new candidate constraint. Using **Theorem 3.2** we translate $\Sigma \cup \{\varphi\}$ into its corresponding set $\Sigma' \cup \{\varphi'\}$ of propositional formulae. Subsequently, we attempt to refute $\Sigma' \cup \{\neg\varphi'\}$. If our completed tableau is closed we can conclude that φ' is already implicitly specified by Σ' . Utilising **Theorem 3.2** once again we can conclude that φ is implied by Σ . Consequently, φ does not need to be specified. In the other case, our completed tableau is open and φ' is not implied by Σ' . In this case, we read off *all* truth assignments that satisfy Σ' and violate φ' . For every model θ of $\Sigma' \cup \{\neg\varphi'\}$ we generate a T -compatible counterexample data tree T'_θ for the implication of φ by Σ . The XML document T'_θ is a two pre-image data tree with respect to v , and W_1, W_2 denote the two pre-images of $T(v)$ in T'_θ . We choose the string values of the two pre-images W_1, W_2 such that they have isomorphic projections to precisely those essential v -subgraphs of T whose corresponding variables are evaluated to *true* by θ . Such a construction is always possible (Hartmann *et al.*, 2008). At this stage the designer may substitute the artificially chosen string values by actual values from the application domain. Note that this does not cause any problems as we only need two distinct values.

Example 5.1: We continue **Example 3.9** and **Example 4.2**. The set of all models for $\Sigma' \cup \{\neg\varphi'\}$ is given by θ_1 and θ_2 in **Example 4.2**. At this stage, the XML designer may choose the XML data trees T'_{θ_1} from the top of Figure 9 to represent θ_1 , and T'_{θ_2} from the bottom of Figure 9 to represent θ_2 , respectively.

After this process, the participants of the acquisition process inspect each of the counterexample trees T' generated from the models of $\Sigma' \cup \{\neg\varphi'\}$, respectively. Each of these sample documents

conforms to the rules represented by Σ and violates the rule represented by φ . If there is at least one such XML document that is relevant for the application domain, then the specification of φ is too restrictive. Consequently, we do not specify φ . In the remaining case, we cannot find any relevant two pre-image data tree with respect to v that satisfies Σ and violates φ . In other words, there is some XML sample document D that satisfies Σ but violates φ , but none of the sample documents with this property are relevant, including D . If we do not specify φ , then D satisfies the constraints in Σ , and is therefore a legal XML document that may occur in our repository. However, D has been as identified as irrelevant for the application domain. Consequently, we must specify φ to constrain our XML repository appropriately.

Example 5.2: For instance, the XML tree T'_{θ_1} in **Example 5.1** is relevant for the application domain. If we now decided to specify φ , then T'_{θ_1} cannot be considered a legal XML document. Hence, the specification of φ would result in the exclusion of reasonable XML documents. Consequently, we do not specify φ .

At the very end of the process, Σ may be redundant in the sense that it contains some Boolean constraint φ that is implied by the remaining constraints in $\Sigma - \{\varphi\}$. However, by testing such implications continuously, we can remove such constraints from Σ until it becomes non-redundant.

Our techniques may become even more powerful when integrated into more general approaches. For instance, the work in Albrecht *et al* (1995) describes a general framework for constraint acquisition combining several techniques such as natural language processing, dialogues, heuristics and sample data. However, in their approach sample data is not generated automatically but entered by the designer. As demonstrated, tableaux techniques can help generating sufficient sample data such that candidate constraints cannot only be discarded, but also be specified.

5.2 SAT Solvers

The main technical problem considered in this paper is the generation of all two-preimage data trees that satisfy all the Boolean constraints in Σ and violate the candidate constraint φ . According to **Theorem 3.2** this problem is equivalent to the generation of all models for the propositional formula that is the conjunction of all formulae in Σ' and the negation of φ' , denoted by $\Lambda\Sigma' \wedge \neg\varphi'$. Recall that the *satisfiability problem of Boolean propositional logic* (SAT) is the problem of deciding whether an arbitrary propositional formula has a model. Consequently, our main technical problem is just the search version of SAT where we enumerate all models.

Propositional tableaux represent only one method for generating all models of a given formula. Alternatively, only modest modifications are generally required to transform any SAT-solver into a tool that produces all satisfying truth assignments (Kautz and Selman, 2007). However, designated algorithms exist that exclusively deal with the search version above (Grumberg *et al*, 2004; Jin and Somenzi, 2005).

In summary, we can choose any algorithm that enumerates all models of the formula $\Lambda\Sigma' \wedge \neg\varphi'$. Any advances that will be made towards improving the performance of current SAT solvers will translate directly into advances in the performance for generating our sample databases.

5.3 Remarks on the Performance

Finally, we make some notes on the time complexity of this approach. In the worst case, we need to generate a number of sample documents that is exponential in the number of different essential v -subgraphs occurring in $\Sigma \cup \{\varphi\}$. For example, the formula $\Lambda\Sigma' \wedge \neg\varphi'$ may be a tautology.

However, enumerating all models of a given formulae can be performed in time proportional to the number of all models and the effort needed to generate each model in isolation (Dechter and Itai, 1992). Even though SAT is the prime representative of an NP-complete problem (Cook, 1971), a substantial body of research shows that nearly all instances of this problem that occur in practice can be solved in a feasible amount of time (Kautz and Selman, 2007).

Moreover, all sample documents contain only two distinct pre-images. According to our approach, we can make the decision about specifying either as soon as we have found a sample document that is relevant for the application, or after we have identified all sample documents as irrelevant.

As pointed out before, it is absolutely crucial for the quality of the target domain to acquire a correct and complete semantics. We believe that it is well worth the time and costs to inspect the candidate constraints carefully at an early stage in the design process. All over, it is a relatively small price to pay when we consider situations in which the entire system must be redesigned because some essential semantic information was not adequately addressed previously.

6. CONCLUSION AND FUTURE WORK

Semantic knowledge acquisition is crucial for the quality of XML design. The existence of appropriate sample data can reduce the problem of acquiring suitable constraints to the problem of assessing the relevance of sample data for the application domain. We have identified off-the-shelf tools for solving problems in propositional logic as a means for generating such an XML sample data collection in the case of Boolean constraints on XML trees.

On the one hand, we would like to extend the expressiveness of our constraints while maintaining decision support for the specification of the extended class. It is also a worthwhile task to analyse Boolean constraints in the presence of other constraint languages, for instance those that can be specified in XML Schema (Thomson *et al*, 2004).

On the other hand, we would like to analyse the trade-off between the expressiveness of different constraint classes and the efficiency of model generation. It may turn out that less expressive constraints suffice in most situations in practice, and for such restricted classes more efficient ways of generating all models may exist. The prime example is the class of functional dependencies which correspond to Horn formulae in propositional logic, and for which implication can be decided in linear time (Hartmann *et al*, 2007).

ACKNOWLEDGEMENT

This research is supported by the Marsden fund council from Government funding, administered by the Royal Society of New Zealand. The first author is supported by a research grant of the Alfried Krupp von Bohlen and Halbach Foundation, administered by the German Scholars Organisation. The third author is supported by a Bright Future Doctoral Scholarship from New Zealand Government funding.

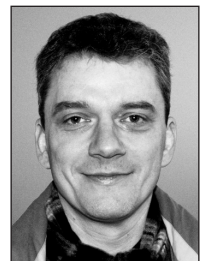
REFERENCES

- ALBRECHT, M., BUCHHOLZ, E., DÜSTERHÖFT, A. and THALHEIM, B. (1995): An informal and efficient approach for obtaining semantic constraints using sample data and natural language processing. In *Semantics in Databases*, of *Lecture Notes in Computer Science*, Springer, 1358: 1–28.
- ARENAS, M. and LIBKIN, L. (2004): A normal form for XML documents. *Trans. Database Syst.* 29(1): 195–232.
- BETH, E. (1959): *The Foundations of Mathematics*. North Holland.
- BRAY, T., PAOLI, J., SPERBERG-McQUEEN, C.M., MALER, E. and YERGEAU, F. (2006): Extensible markup language (XML) 1.0 (fourth edition) W3C recommendation, August. <http://www.w3.org/TR/xml/>

- BUNEMAN, P., FAN, W., SIMÉON, J. and WEINSTEIN, S. (2001): Constraints for semi-structured data and XML. *SIGMOD Record* 30(1): 47–54.
- COOK, S. (1971): The complexity of theorem-proving procedures. In *ACM Symposium on Theory of Computing*, 151–158.
- DECHTER, R. and ITAI, A. (1992): Finding all solutions if you can find one. Tech. rep., In AAAI-92 Workshop on Tractable Reasoning.
- DEMETROVICS, J. (1980): On the equivalence of candidate keys with sperner systems. *Acta Cybern.* 4: 247–252.
- DEMETROVICS, J., RONYAI, L. and SON, H. (1991): Dependency types. *Computers Math. Applic.* 21(1): 25–33.
- DEMETROVICS, J., RONYAI, L. and SON, H. (1991): On the representation of dependencies by propositional logic. In *MFDBS 1991 of Lecture Notes in Computer Science*, Springer 495: 230–242.
- ENDERTON, H. (2001): *A mathematical introduction to logic: Second Edition*. Academic Press,.
- FAGIN, R. (1982): Armstrong databases. Tech. Rep. RJ3440(40926), IBM Research Laboratory, San Jose, California, USA.
- FAN, W. (2005): XML constraints. In *DEXA Workshops*, 805–809.
- FAN, W. and SIMÉON, J. (2003): Integrity constraints for XML. *J. Comput. Syst. Sci.* 66(1): 254–291.
- GENTZEN, G. (1935): Untersuchungen über das logische Schließen. *Mathematische Zeitschrift* 39: 176–210, 405–431.
- GRUMBERG, O., Schuster, A. and YADGAR, A. (2004): Memory efficient all-solutions SAT solver and its application for reachability analysis. In *5th International Conference on Formal Methods in Computer-Aided Design in Lecture Notes in Computer Science*, Springer, 3312: 275–289.
- HARTMANN, S. and LINK, S. (2003): More functional dependencies for XML. In *AdBIS in Lecture Notes in Computer Science*, Springer, 2798: 355–369.
- HARTMANN, S. and LINK, S. (2007): Unlocking keys for XML trees. In *ICDT in Lecture Notes in Computer Science*, Springer, 4353: 104–118.
- HARTMANN, S. and LINK, S. (2008): Characterising nested data dependencies by fragments of propositional logic. *Annals of Pure and Applied Logic* 152(1–3): 84–106.
- HARTMANN, S., LINK, S. and TRINH, T. (2007): Efficient reasoning about XFDs with pre-image semantics. In *DASFAA 2007 in Lecture Notes in Computer Science*, Springer, 4443: 1070–1074.
- HARTMANN, S., LINK, S. and TRINH, T. (2008): Boolean constraints for XML modeling. *Proceedings of the European-Japanese Conference on Conceptual Modeling*.
- HARTMANN, S. and TRINH, T. (2006): Axiomatising functional dependencies for XML with frequencies. In *FoIKS in Lecture Notes in Computer Science*, Springer, 3861: 159–178.
- HINTIKKA, K. (1955): Form and content in quantification theory. *Acta Philosophica Fenica*, 8: 7–55.
- JIN, H. and SOMENZI, F. (2005): Prime clauses for fast enumeration of satisfying assignments to boolean circuits. In *Proceedings of the 42nd Design Automation Conference*, 750–753.
- JUNGNICKEL, D. (1999): *Graphs, Networks and algorithms*. Springer.
- KATONA, G. and TICHLER, K. (2006): Some contributions to the minimum representation problem of key systems. In *FoIKS of Lecture Notes in Computer Science*, Springer, 3861: 240–257.
- KAUTZ, H. and SELMAN, B. (2007): The state of SAT. *Discrete Applied Mathematics* 155(12): 1514–1524.
- LEE, M., LING, T. and LOW, W. (2002): Designing functional dependencies for XML. In *EDBT in Lecture Notes in Computer Science*, Springer, 2287: 124–141.
- SAGIV, Y., DELOBEL, C., PARKER Jr., D.S. and FAGIN, R. (1981): An equivalence between relational database dependencies and a fragment of propositional logic. *J. ACM* 28(3): 435–453.
- SMULLYAN, R. (1995): *First-order Logic*. Dover Publications.
- THOMPSON, H., BEECH, D., MALONEY, M. and MENDELSON, N. (2004): XML Schema part 1: Structures second edition, W3C Recommendation, October. <http://www.w3.org/TR/REC-xmlschema-1-20041028/>
- VINCENT, M., LIU, J. and LIU, C. (2004): Strong functional dependencies and their application to normal forms in XML. *ACM Trans. Database Syst.* 29(3): 445–462.

BIOGRAPHICAL NOTES

Sven Hartmann received his PhD in 1996 and his DSc in 2001, both from the University of Rostock (Germany). Currently, he is professor and chair of Databases and Information Systems at Clausthal University of Technology (Germany). He was professor and deputy director of the Information Science Research Centre at Massey University (New Zealand) until 2007. His research interests include database systems, conceptual modelling, optimal discrete structures and algorithms, service computing and bioinformatics.



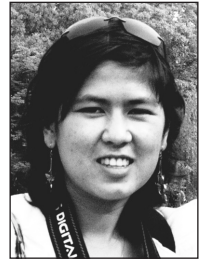
Sven Hartmann

Sebastian Link received his PhD from Massey University (New Zealand) in 2005. Currently, he is an associate professor at the School of Information Management, The Victoria University of Wellington (New Zealand). He is also a member of The Centre for Logic, Language and Computation at The Victoria University of Wellington. His research interests include data modelling, database design, database security, and database theory.



Sebastian Link

Thu Trinh received her MSc from Massey University (New Zealand) in 2005. Currently, she is studying towards a PhD in information systems. Her research interests include database concepts, XML and multimedia.



Thu Trinh