Modeling and Manipulating the Structure of Portal Catalogs

Theodore Dalamagas, Alexandra Meliou, Timos Sellis

Knowledge and Database Systems Lab
School of Electrical and Computer Engineering
National Technical University of Athens
Zographou 157 73, Athens, Hellas
{dalamag,ameliou,timos}@dblab.ece.ntua.gr

Abstract

Portal catalog sites are Web information nodes that provide querying and browsing capabilities on data organized in a thematic hierarchy. Up to now, portal catalogs have not been treated as full-fledged objects so as to allow for their manipulation. A major challenge is thus to complement catalog search and browsing with operations that manipulate structural information from similar vertical portal catalog sites, that is sites of a specific subject or domain.

This work proposes models and operators that allow for structural manipulation of vertical portal catalog sites. First, we explore the algebraic properties of trees representing hierarchies of portal catalogs, and define a lattice algebraic structure on them. Then, turning this structure into a boolean algebra, we present the operators S-union, S-intersection and S-difference to support structural manipulation of such trees, we give the laws that hold for these operators and we identify the conditions under which this framework is applicable.

1. Introduction

The Web information space is a set of information nodes. Each information node maintains a data source, a data schema, a query engine and a query interface. The data source serves as the storage place of data. The data schema describes the content of the data source using concept/role or entity/relationship or semistructured models. The query engine implements the query mechanism, and, finally, the query interface provides query capabilities for users. Portal catalogs are Web information nodes that provide querying and browsing capabilities on data organized in a hierarchy, on a category/subcategory basis.

New factors have changed the requirements for an efficient information retrieval in the Web information space:

- There are many *vertical* portal catalogs, that is catalogs of a specific subject or domain (e.g. e-marketplaces for hardware, photo equipment, cultural resources, etc). These sites provide better querying capabilities compared to the traditional internet search engines: keyword search, category browsing, category search and sometimes SQL-like querying.
- There is a huge amount of information which can be retrieved in the form of dynamic-generated Web pages due to the high usage of database systems for storing data. Such information is often stored in many data sources with similar (but not identical) schemas.

Taking the above into consideration, effective information retrieval is the process which supports the identification and usage of Web information nodes that provide querying capabilities for data relevant to an information need.

Portal catalog sites are quite useful in such retrieval tasks, since they maintain large volumes of information resources which is not possible for a single user to classify and exploit. However, up to now, portal catalogs have not been treated as full-fledged objects so as to allow for their manipulation. Moreover, the XML infrastructure has become a popular means for enabling automatic processing of Web information nodes. Given that XML data are organized in a hierarchically structured and self-describing manner, there is a need that the structural aspect should receive stronger attention. There is a major challenge to complement catalog search and browsing with operations that manipulate structural information from similar vertical portal catalogs, that is a set of portal catalogs providing querying and browsing capabilities on a specific subject or domain. Such a set of portal catalogs can be a powerful tool to assist the retrieval task in the Web information space. The next section clarifies such a motivation.

1.1. Motivating example

Adorama, B&H and RitzCamera[1] are three e-marketplaces for photo equipment. The first two maintain a complex hierarchy with categories/subcategories to assist searching and browsing for their products, while the last one provides a simple one-level hierarchy to categorize products and only browsing capabilities. Figure 1 shows parts of their hierarchy. Notice that a schema matching pre-processing (see also related work in Section 1.3) helps the identification of all matching categories (nodes) from both catalogs, since there may be categories with different labels, although they are semantically similar. Where matching is not straightforward due to different naming, we provide the necessary information giving the matching categories in Figures 1, 2 and 3.

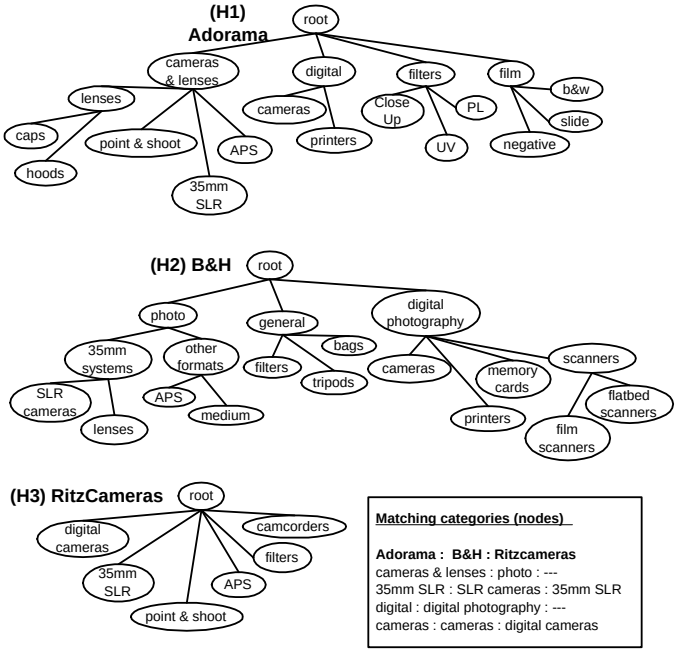

Figure 1. Part of Adorama's, B&H's and RitzCamera's hierarchy.

One can think of various query operations on data provided by those catalogs:

1. Browse the hierarchies to find 35mm SLR cameras in all catalogs.

2. Pose path expression queries on Adorama's hierarchy like `/Filters/UV/"price<40"`, that is find ultra-violet filters with price less than 40euros.

[1] www.adorama.com, www.bhphotovideo.com, www.ritzcamera.com

However, looking at the three e-marketplaces as *a set of similar vertical portal catalogs* maintaining resources relevant to photo equipment, there is a need to complement such kind of querying with operations that manipulate structural information from the hierarchies of these catalogs. Such structural information is quite important since it acts as a semantic guide for query operations on data. Under this perspective, one can think of various query operations on structural information provided by portal catalogs:

1. Merge portal catalogs to see which is the integrated structural information provided by their hierarchies: Figure 2 presents a merged catalog from Adorama's and B&H catalogs. In this new catalog, there are categories that were only in Adorama's catalog, or only in B&H's catalog, or in both. Notice that the structural relationships between the categories are preserved in this new catalog where possible: an ancestor/descendant relationship is always preserved, while a parent/child relationship may be changed to an ancestor/descendant one. In Figure 3, the catalog produced in Figure 2 is

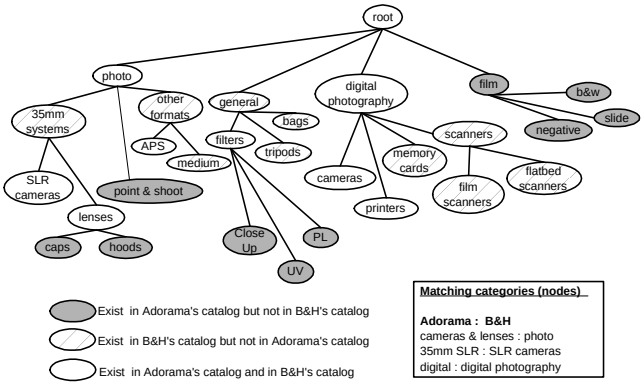

Figure 2. Merging Adorama's and B&H's catalogs.

merged with RitzCameras' catalog.

2. Find the common part of portal catalogs: Figure 4 presents the catalog which is common in both Adorama's and B&H catalogs. Notice that the structural relationships between the categories are preserved in this new catalog where possible: a parent/child relationship is always preserved, while an ancestor/descendent relationship may be changed to a parent/child one.

3. Find the part of a portal catalog which is not present in another catalog: Figure 5 presents the part of Adorama's catalog which is not present in B&H's catalog.

4. Find the common part of two portal catalogs and merge it with another catalog: Figure 6 presents the catalog

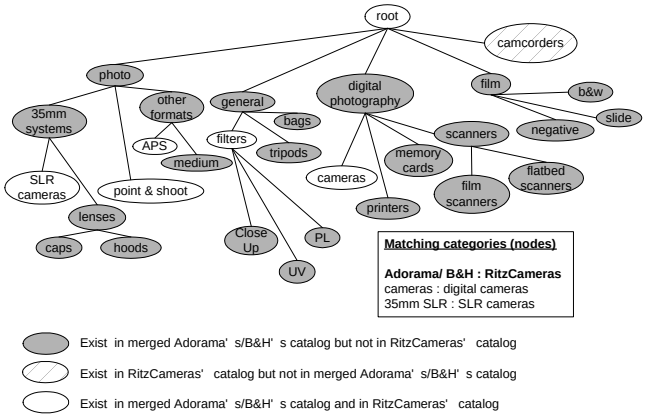

Figure 3. Merging Adorama/B&H's and RitzCameras' catalogs.

produced by merging RitzCamera's catalog with the common part of both Adorama's and B&H catalogs.

Several issues arise for such kind of query operations. Among them, we note:

1. the need to provide a theoretical framework to support such operations, that is a framework to support structural manipulation of portal catalogs,
2. the detection of structural inconsistencies that might prevent such kind of operations,
3. the utilization of schema matching techniques to identify matching categories in different catalogs, that is categories which are semantically similar although their label might be different.

1.2. Contribution

This work proposes models and operators that allow for structural manipulation of vertical portal catalog sites. Our main contributions are:

1. We present a tree-like model for hierarchies of portal catalog sites.

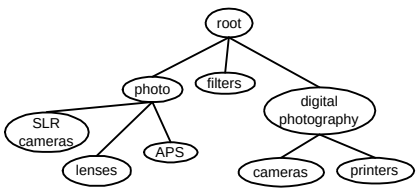

Figure 4. The common part of Adorama's and B&H's catalogs.

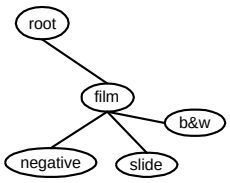

Figure 5. The part of Adorama's catalog which is not present in B&H's catalog.

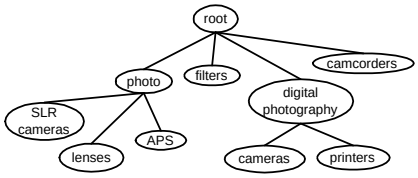

Figure 6. Merging RitzCamera's catalog with the common part of both Adorama's and B&H catalogs.

2. We explore the algebraic properties of trees representing hierarchies of portal catalogs, and define a lattice algebraic structure on them.
3. Turning this structure into a boolean algebra, we present the S-union operator to get an integrated form of all structural information from the involved trees, the S-intersection operator to get all common structural information from the involved trees, and, using complement trees, the S-difference operator to get structural information present in one tree and not in the other.
4. Finally, we give the laws that hold for these operators and we identify the conditions under which this framework is applicable.

1.3. Related work

Related issues have been exploited in research areas like schema merging and integration, ontology construction, semistructured data management and complex object management.

Schema merging and information integration is a major subject in the area of Database Systems. Much work has been done on building an integrated schema from heterogeneous sources using a common data model, especially using some variant of the ER model ([18], [22], [12], [8], [32], [23]). Theoretical aspects of schema merging have been discussed in [5], [27]. Finally, finding similarities between objects of different schemas and dealing with semantic conflicts to support schema matching is crucial for schema merging and information integration ([16], [33],

[34], [15], [28], [26], [30], [21]). Most recent research focuses on ontologies ([36], [13], [35]), where methods for ontology composition and merging are presented, and XML schemas trees ([25], [4], [20], [6]), where methods to unify and integrate trees representing XML data are discussed. Also, the identification of common tree-patterns to enhance the performance of join queries on XML data have received strong attention ([10], [6], [2]). Finally, general schema definition and structural querying capabilities are provided from RDF/XML schema languages, like for example RQL ([14], [19]). However, the presented related work is either focused on the detection and resolving of semantic conflicts to support the schema integration task in the presence of strict typed and semantically rich schemas, or puts emphasis only on structural transformation and querying of schemas. Our work, on the other hand, is based on the need to provide an integrated formal framework for structural manipulation on hierarchical schemas with lightweight semantics, in the form of query operators on trees to support set-like operations (union, intersection and difference).

In this sense, prior research on complex object management is closely related to our work. Complex objects propose nested relations against normalized ones in the attempt to overcome the restrictions imposed by flat relations. [3] presents a calculus for complex objects. A lattice structure is defined on nested objects and operators on these objects are introduced. However, the operators are used to provide an extension of Horn clauses as a calculus to define formally a path-expression query language on structures. Neither complements are suggested nor a difference operator is defined. Operation and implementation issues for complex objects are presented in [17]. Again the target is to to modify and query the configuration of a complex object. A join operator is introduced, similar to the relational join, but neither union, nor intersection, nor difference operators are used. [29] suggests an extension of the relational algebra to capture complex objects. Other works, like [31], model complex objects with object-oriented schemas, and introduce class union using attribute merging, class intersection using attribute matching and class difference using missing attributes. Generally, the main focus of the related work in complex objects is on how to select and reconstruct substructures of the original structures, as one can see in [1], and on supporting structural manipulation on the basis of either merging structures or finding their common parts or their different parts.

1.4. Outline

Section 2 deals with modeling issues for representing hierarchies of portal catalogs. Sections 3 explores the algebraic properties of trees representing portal catalog hierarchies and introduces the S-union and S-intersection operators. Section 4 defines a lattice algebraic structure on these trees, and turns this structure into a boolean algebra, introducing the S-difference operator and a set of laws hold. Section 5 discusses further the presented framework. Section 6 describes an application example and, finally, Section 7 concludes this paper.

2. Representing hierarchies for portal catalogs

Portal catalogs provide data organized in hierarchies, on a category/subcategory basis. Such kind of hierarchies can be seen as lightweight ontologies, with their concepts (categories) related with primitive semantics *isA* and *hasA*, a simplified version of data abstraction mechanisms discussed in the semantic data modeling literature [11, 9]. *isA* represents the supertype/subtype relationship, while *hasA* refers to aggregation or association. For example in `Arts/(Painting,Dance)`, `Arts` is a supertype while *Painting* and *Dance* are subtypes, Also, in `Computer/(Hardware,Software)`, `Computer` is an aggregation of `Hardware` and `Software`, that is a relationship between objects is considered as a higher level object. Finally, in `Computer/Companies`, `Companies` is an association of `Computers`, that is a collection of objects is considered as a higher level object.

Faceted classification is another aspect of portal catalogs hierarchies[2]. Data provided by portal catalogs can be classified in *facets*. For example `range of prices`, `geographical areas`, `product types`, etc are facets. `low-price`, `mid-price`, `high-price` are the *headings* of the facet `range of prices`, while `new`, `second hand` are the headings of the facet `product type`. Figure 9 shows a hierarchy that has the categorization `new` and `second hand`, headings of the facet `product type`, as a top level choice during browsing. Other headings could be used, combined with the headings `new` and `second hand`, to further expand the hierarchy, for example `low-price`, `mid-price`, `high-price`. Facet headings can form many different hierarchies in the form of tree structures, that alone or combined can classify data provided by a portal catalog, depending on the user's need.

However, the user is neither aware of the semantics provided nor of the different facets used to construct the hierarhcy. What is available in a portal catalog is usually a tree-like structured hierarchy that will assist her during a browsing retrieval task.

The construction of a such a tree-like structured hierarchy for a portal catalog can be based on a *global* catalog available for the specific application domain or community. Figure 7 presents an example of such a catalog S for photo

[2] see XFML (www.xfml.org) and Topic Maps (www.topicmaps.org)

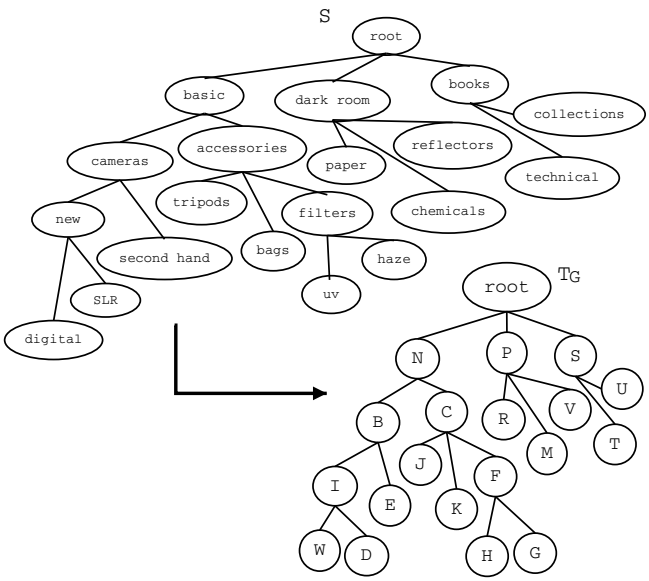

Figure 7. A global catalog S for photo equipment and its representative tree T_G.

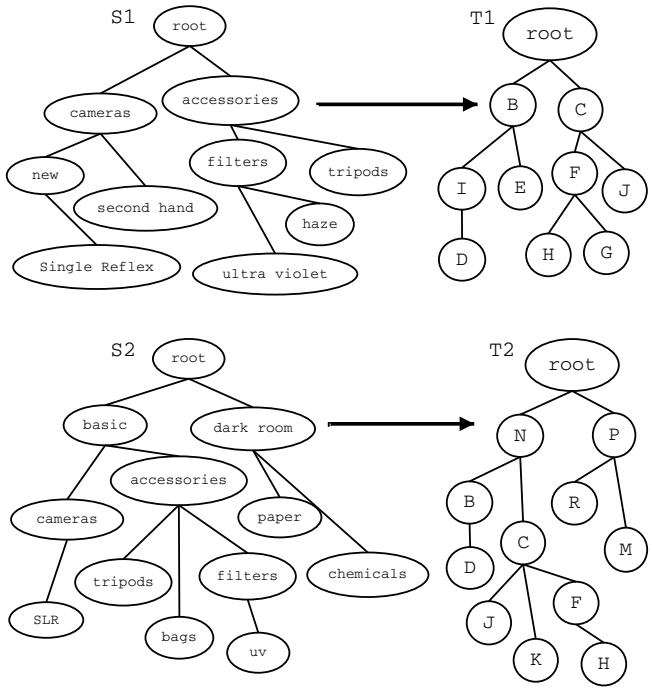

Figure 8. Catalogs S_1, S_2 and their representative trees T_1 and T_2.

equipment. Figure 8 shows two catalogs S_1 and S_2 based on global catalog S. In such case, portal catalogs are actually parts of this global catalog and may have only inconsistencies related to naming resolving, but not structural inconsistencies. For example, in Figure 8, `Single Reflex` and `ultra violet` in S_1 are `SLR` and `uv` in S_2. On the other hand, if such a global catalog is not available, structural inconsistencies may occur. See for example the catalog S_3 in Figure 9. The author of S_3 prefers having the categorization `new` and `second hand` as a top level choice during browsing, which is inconsistent with S_1.

We start exploring the algebraic properties of trees representing hierarchies of portal catalogs, based on the assumption that there is a catalog available for the specific application domain or community. Then, we provide a framework to ensure that similar algebraic properties hold without having such a global catalog.

3. Algebraic properties of portal catalog hierarchies

We represent hierarchies for portal catalogs as rooted labeled trees. Such trees may be the output of schema matching tools that identify semantic similarities and resolve conflicts (see Section 1.3). Figures 7, 8 show examples of such trees that we will use throughout this work as presentation examples. We note that similar labels are used for nodes in one catalog that match nodes of another catalog.

Let $N = \{n_1, n_2, \ldots, n_k\}$ a set of nodes and r the *root* node. A function $label(x)$ assigns a string label as unique identifier for every node $x \in N$, while $label(r) =$`root`. Node A is the *parent* of node B, $p(A, B)$, if there is a direct edge from A to B. Node A is an *ancestor* of node B, $a(A, B)$, if there is a direct path of k edges, $k > 1$, from A to B.

Definition 3.1 *A rooted labeled tree, or just 'tree' from now on, T is the set $\{r\} \cup N$ equipped with the relation $\mathcal{P}$: $\langle \{r\} \cup N, \mathcal{P} \rangle$. $x\mathcal{P}y$ holds if there is a direct edge from node x to node y ($x, y \in N$), that is node x is the parent of node y, $p(x, y)$. Especially for the root node r, $\exists y : r\mathcal{P}y$.*

Two trees are equal if they have the same nodes and the same structural relationships among them, as the following definition shows.

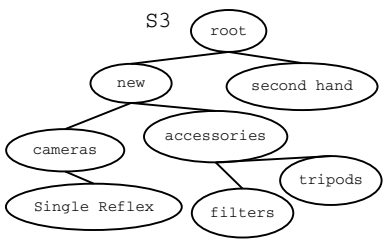

Figure 9.

Definition 3.2 *Trees T_i and T_j are equal, $T_i = T_j$, iff $N_i = N_j$ and $\forall\, x, y\, with\, x\mathcal{P}_i y$ then $x\mathcal{P}_j y$.*

$\mathcal{P}^{tr}$ denotes the transitive closure of $\mathcal{P}$, that is $x\mathcal{P}^{tr}y$ holds if for any elements x and y of $\{r\} \cup N$ there exist $c_0, c_1, \ldots, c_n$ with $c_0 = x$, $c_n = y$ and $c_p \mathcal{P} c_{p+1}$ for all $0 \leq p < n$.

We now define the S-subset, $\subseteq^s$, relation for 2 trees. Intuitively, when $T_1 \subseteq^s T_2$, all nodes of T_1 exist in T_2 and the structural relationships in T_1 which involve nodes present in T_2 are maintained in T_2.

Definition 3.3 *The tree T_0 with only one node (its root) is an S-subset, $\subseteq^s$, of any tree. A tree T_i, $\langle \{r\} \cup N_i, \mathcal{P}_i \rangle$, is an S-subset of a tree T_j, $\langle \{r\} \cup N_j, \mathcal{P}_j \rangle$, or $T_i \subseteq^s T_j$, if $\forall\, x, y\, with\, x\mathcal{P}_i y$ then $x\mathcal{P}_j^{tr} y$.*

For example, in Figure 10, $T_1 \subseteq^s T_3$, while $T_2 \not\subseteq^s T_3$ since although $E\mathcal{P}_2 C$ holds in T_2, $E\mathcal{P}_3^{tr} C$ does not hold in T_3.

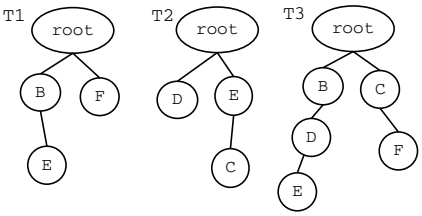

Figure 10. $T_1 \subseteq^s T_3, T_2 \not\subseteq^s T_3$.

Lemma 3.1 *Let T_i, $\langle \{r\} \cup N_i, \mathcal{P}_i \rangle$, and T_j, $\langle \{r\} \cup N_j, \mathcal{P}_j \rangle$ be two trees. If $N_i = N_j$ and $T_i \subseteq^s T_j$ then $T_i = T_j$.*

Proof: Directly from Definitions 3.3 and 3.2.

□

Let T_G, $\langle \{r\} \cup N_G, \mathcal{P}_G \rangle$, be a tree which we call *global tree* and $\mathcal{S}_G = \{T_i : T_i \subseteq^s T_G\}$, that is all trees which are S-subsets of T_G. Note that $\mathcal{S}_G$ includes T_0 $\langle \{r\}, \mathcal{P} \rangle$, which is the tree with only one node (its root), and T_G. For example, trees T_1 and T_2 in Figure 8 are S-subsets of T_G in Figure 7: $T_1 \subseteq^s T_G$ and $T_2 \subseteq^s T_G$. Using Theorem 3.1, we claim that the set $\mathcal{S}_G$ equipped with the relation $\subseteq^s$, $\langle \mathcal{S}_G, \subseteq^s \rangle$, is a partial order.

Theorem 3.1 *The $\subseteq^s$ relation on $\mathcal{S}_G$, $\langle \mathcal{S}_G, \subseteq^s \rangle$, is*

1. *reflexive: $T_i \subseteq^s T_i$,*
2. *antisymmetric: $T_i \subseteq^s T_j$ and $T_j \subseteq^s T_i$ imply $T_i = T_j$, and*
3. *transitive: $T_i \subseteq^s T_j$ and $T_j \subseteq^s T_k$ imply $T_i \subseteq^s T_k$.*

thus $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a partial order.

Proof: Reflexivity holds since when $T_i \mathcal{P} T_i$ holds then $T_i \mathcal{P}^{tr} T_i$ holds, too. Antisymetry holds: if $T_i \subseteq^s T_j$ and $T_j \subseteq^s T_i$, then $N_i \subseteq N_j$ and $N_j \subseteq N_i$, and, thus, $N_i = N_j$. Given $T_i \subseteq^s T_j$ and $N_i = N_j$, $T_i = T_j$ holds (Lemma 3.1). Transitivity holds, too. If $T_i \subseteq^s T_j$ and $T_j \subseteq^s T_k$, then

$$\forall\, x, y\, with\, x\mathcal{P}_i y \Rightarrow x\mathcal{P}_j^{tr} y \quad (1)$$

$$\forall\, x, y\, with\, x\mathcal{P}_j y \Rightarrow x\mathcal{P}_k^{tr} y \quad (2)$$

From (1), for all x and y of $\{r\} \cup N_i$ there exist $c_0, c_1, \ldots, c_n$ with $c_0 = x$, $c_n = y$ and $c_p \mathcal{P}_i c_{p+1}$ for all $0 \leq p < n$. For $n = 1$, $x\mathcal{P}_j y$ holds, and, thus, (2) gives $x\mathcal{P}_k^{tr} y$. For $n > 1$, $x\mathcal{P}_j c_1$, $c_1 \mathcal{P}_j c_2$, $\ldots$, $c_{n-1}\mathcal{P}_j y$ hold, thus, $x\mathcal{P}_k^{tr} c_1$, $c_1 \mathcal{P}_k^{tr} c_2$, $\ldots$, $c_{n-1}\mathcal{P}_k^{tr} y$ hold, and this gives $x\mathcal{P}_k^{tr} y$.

□

We note that the partial order $\langle \mathcal{S}_G, \subseteq^s \rangle$ has T_0 as its bottom element $\perp$, with the property that $\perp \subseteq^s T_i$, and T_G as its top element $\top$, with the property that $T_i \subseteq^s \top$, for all $T_i \in \mathcal{S}_G$.

We next define the first 2 operators for structural reasoning on trees in representing hierarchies for portal catalogs: *S-union* and *S-intersection*. Both operators are binary, in the sense that they get two trees as input and produce a new tree.

3.1. S-union ($\cup^s$)

Intuitively, the *S-union* of two trees T_i and T_j, $T_i \cup^s T_j$, is the tree which provides integrated structural information from T_i and T_j. The structural relationships between the nodes are preserved in this new tree where possible: an ancestor relationship is always preserved, while a parent relationship may be changed to an ancestor one. Figure 11 presents an example of an S-union operation on trees T_1 and T_2, both S-subsets of T_G (Figure 7), resulting to the tree $T_3 = T_1 \cup^s T_2$. The definition for the S-union operation follows.

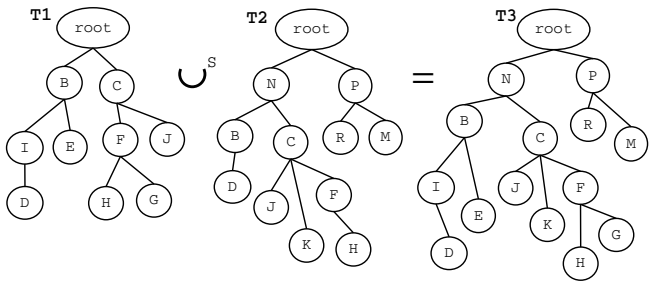

Figure 11. Union operation: $T_3 = T_1 \cup^s T_2$.

Definition 3.4 *Let T_i, $\langle \{r\} \cup N_i, \mathcal{P}_i \rangle$, and T_j, $\langle \{r\} \cup N_j, \mathcal{P}_j \rangle$, be two trees in $\mathcal{S}_G$. The S-*union *of T_i and T_j,*

$T_i \cup^s T_j$, is the tree T, $\langle \{r\} \cup N, \mathcal{P} \rangle$, constructed as follows:

- $N = N_i \cup N_j$:
  T has all nodes from both T_i and T_j.
- $x\mathcal{P}y$, *if* $x\mathcal{P}_G y$:
  T keeps all parent structural relationships $p(x, y)$ from T_G that involve nodes x, y present in either T_i or T_j ($x, y \in N$).
- $x\mathcal{P}y$, *if* $\neg x\mathcal{P}_G y$ *and* $x\mathcal{P}_G^{tr} y$ *and all* $c_1, c_2, \ldots, c_{n-1}$ *in* $x\mathcal{P}_G c_1$, $c_1\mathcal{P}_G c_2$, $\ldots$, $c_{n-1}\mathcal{P}_G y$, $n \geq 2$, *are not in* N:
  T uses as parent relationships all ancestor relationships $a(x, y)$ from T_G that involve nodes x, y present in either T_i or T_j ($x, y \in N$), in the path of which all nodes belong neither to T_i nor to T_j.

We note that

1. $(T_i \cup^s T_j) \subseteq^s T_G$ since $x\mathcal{P}_G^{tr} y$ for every $x\mathcal{P}y$, thus $(T_i \cup^s T_j) \in \mathcal{S}_G$.
2. $(T_i \cup^s T_j)$ is an *upper bound* of $\{T_i, T_j\} \subset \mathcal{S}_G$, i.e. $T_i \subseteq^s (T_i \cup^s T_j)$ and $T_j \subseteq^s (T_i \cup^s T_j)$, since by the way we construct $T_i \cup^s T_j$, it keeps all parent relationships $p(x, y)$ from T_G that involve nodes x, y present either in T_i or in T_j.

3.2. S-intersection ($\cap^s$)

Intuitively, the *S-intersection* of two trees T_i and T_j, $T_i \cap^s T_j$, is the tree which provides structural information common in T_i and T_j. The structural relationships between the nodes are preserved in this new tree where possible: a parent relationship is always preserved, while an ancestor relationship may be changed to a parent one. Figure 12 presents an example of an S-intersection operation on trees T_1 and T_2, both S-subsets of T_G (Figure 7), resulting to the tree $T_3 = T_1 \cap^s T_2$. The definition for the S-intersection

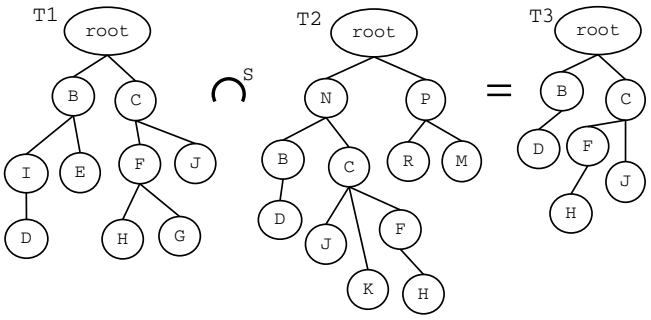

Figure 12. Intersection operation: $T_3 = T_1 \cap^s T_2$.

operation follows.

Definition 3.5 *Let T_i, $\langle \{r\} \cup N_i, \mathcal{P}_i \rangle$, and T_j, $\langle \{r\} \cup N_j, \mathcal{P}_j \rangle$, be two trees in $\mathcal{S}_G$. The* S-intersection *of T_i and T_j, $T_i \cap^s T_j$, is the tree T, $\langle \{r\} \cup N, \mathcal{P} \rangle$, constructed as follows:*

- $N = N_i \cap N_j$:
  T has all nodes common in T_i and T_j.
- $x\mathcal{P}y$, *if* $x\mathcal{P}_G y$:
  T keeps all parent structural relationships $p(x, y)$ from T_G that involve nodes x, y present in both T_i and T_j ($x, y \in N$).
- $x\mathcal{P}y$, *if* $\neg x\mathcal{P}_G y$ *and* $x\mathcal{P}_G^{tr} y$ *and all* $c_1, c_2, \ldots, c_{n-1}$ *in* $x\mathcal{P}_G c_1$, $c_1\mathcal{P}_G c_2$, $\ldots$, $c_{n-1}\mathcal{P}_G y$, $n \geq 2$, *are not in* N:
  T uses as parent relationships all ancestor relationships $a(x, y)$ from T_G that involve nodes x, y present in both T_i and T_j ($x, y \in N$), in the path of which all nodes do not belong to the set of common nodes of T_i and T_j.

We note that

1. $(T_i \cap^s T_j) \subseteq^s T_G$ since $x\mathcal{P}_G^{tr} y$ for every $x\mathcal{P}y$, thus $(T_i \cap^s T_j) \in \mathcal{S}_G$.
2. $(T_i \cap^s T_j)$ is a *lower bound* of $\{T_i, T_j\} \subset \mathcal{S}_G$, i.e. $(T_i \cap^s T_j) \subseteq^s T_i$ and $(T_i \cap^s T_j) \subseteq^s T_j$, since by the way we construct $T_i \cap^s T_j$, it keeps only the parent structural relationships $p(x, y)$ from T_G that involve nodes x, y present in both T_i and T_j.

4. Portal catalog hierarchies as algebraic structures

This section goes further by using the $\cup^s$ and $\cap^s$ operators to show that the partial order $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a lattice and keeps the properties of a boolean algebra.

In the previous Section we showed that $\cup^s$ gives an upper bound and $\cap^s$ gives a lower bound of the two trees from $\mathcal{S}_G$ involved in these two operations. As Theorem 4.1 shows, $\cup^s$ gives the *least upper bound*, i.e. the upper bound which is an S-subset, $\subseteq^s$, of all upper bounds of the two trees involved, while $\cap^s$ gives the *greatest lower bound*, i.e. the lower bound which all lower bounds of the two trees involved are S-subset, $\subseteq^s$, of.

Theorem 4.1 *Let T_i, $\langle \{r\} \cup N_i, \mathcal{P}_i \rangle$, and T_j, $\langle \{r\} \cup N_j, \mathcal{P}_j \rangle$, be two trees in $\mathcal{S}_G$. The set of all upper bounds of $\mathcal{S}_G$ is $\mathcal{S}_G^u = \{T_k \in \mathcal{S}_G : T_i \subseteq^s T_k \; and \; T_j \subseteq^s T_k\}$. The set of all lower bounds of $\mathcal{S}_G$ is $\mathcal{S}_G^l = \{T_k \in \mathcal{S}_G : T_k \subseteq^s T_i \; and \; T_k \subseteq^s T_j\}$. Then*

1. *$(T_i \cup^s T_j) \subseteq^s T_p$, for all T_p in $\mathcal{S}_G^u$, that is $T_i \cup^s T_j$ gives the least upper bound of T_i and T_j,*

2. $T_p \subseteq^s (T_i \cap^s T_j)$, *for all* T_p *in* $\mathcal{S}_G^l$, *that is* $T_i \cap^s T_j$ *gives the greatest lower bound of* T_i *and* T_j.

Proof: Let T be a tree for which $T \subseteq^s (T_i \cup^s T_j)$, $T_i \subseteq^s T$ and $T_j \subseteq^s T$. T should have either less nodes than $T_i \cup^s T_j$, or the same number of nodes. In the first case, either $T_i \not\subseteq^s T$ or $T_j \not\subseteq^s T$. In the second case, $T = T_i \cup^s T_j$ (Lemma 3.1). So there is not such a tree T other than $T_i \cup^s T_j$.

Let now T be a tree for which $(T_i \cap^s T_j) \subseteq^s T$, $T \subseteq^s T_i$ and $T \subseteq^s T_j$. T should have either more nodes than $T_i \cap^s T_j$, or the same number of nodes. In the first case, either $T \not\subseteq^s T_i$ or $T \not\subseteq^s T_j$. In the second case, $T = T_i \cap^s T_j$ (Lemma 3.1). So there is not such a tree T.

□

Since the least upper bound and the greatest lower bound exist for all trees T_i, T_j in the partial order on $\mathcal{S}_G$ equipped with the relation $\subseteq^s$, $\langle \mathcal{S}_G, \subseteq^s \rangle$, then $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a lattice.

Theorem 4.2 *The* $\mathcal{S}_G$ *equipped with the relation* $\subseteq^s$, $\langle \mathcal{S}_G, \subseteq^s \rangle$ *is a lattice.*

Proof: Directly from Theorems 3.1 and 4.1

□

Since $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a lattice, the following laws hold [7]:

- Idempotency laws
  1. $T_i \cap^s T_i = T_i$
  2. $T_i \cup^s T_i = T_i$
- Commutative laws
  1. $T_i \cap^s T_j = T_j \cap^s T_i$
  2. $T_i \cup^s T_j = T_j \cup^s T_i$
- Associative laws
  1. $T_i \cap^s (T_j \cap^s T_k) = (T_i \cap^s T_j) \cap^s T_k$
  2. $T_i \cup^s (T_j \cup^s T_k) = (T_i \cup^s T_j) \cup^s T_k$
- Absorption laws
  1. $T_i \cap^s (T_i \cup^s T_j) = T_i$
  2. $T_i \cup^s (T_i \cap^s T_j) = T_i$

Next, we introduce laws linking $\cap^s$ and $\cup^s$, turning the lattice into a *distributive* one.

- Distributive laws
  1. $T_i \cup^s (T_j \cap^s T_k) = (T_i \cup^s T_j) \cap^s (T_i \cup^s T_k)$
  2. $T_i \cap^s (T_j \cup^s T_k) = (T_i \cap^s T_j) \cup^s (T_i \cap^s T_k)$

We note (see [7]) that for any a, b, c in a lattice $\langle L, \geq \rangle$, with $\vee$ denoting the least upper bound and $\wedge$ the greatest lower bound,

$$a \wedge (b \vee c) \geq (a \wedge b) \vee (a \wedge c) \quad (3)$$

and dually

$$a \vee (b \wedge c) \geq (a \vee b) \wedge (a \vee c) \quad (4)$$

Since the equality does not hold in general, the distributive laws should be proved. Theorem 4.3 shows that the distributive laws hold, thus $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a distributive lattice.

Theorem 4.3 *The* $\mathcal{S}_G$ *equipped with the relation* $\subseteq^s$, $\langle \mathcal{S}_G, \subseteq^s \rangle$, *is a distributive lattice.*

Proof: Omitted due to lack of space. The complete proof is presented in [24].

□

We next provide $\langle \mathcal{S}_G, \subseteq^s \rangle$ with additional structure to support *complements*.

4.1. Complements

Intuitively, the complement of a tree $T_i \in \mathcal{S}_G$ is the tree $T_i' \in \mathcal{S}_G$ which provides structural information present in T_G and not in T_i. Given T_i, T_i', there must be two trees T_j, $T_k \in \mathcal{S}_G$, denoted by $\mathbf{0}$ and $\mathbf{1}$, respectivelly, so that

1. $T_i \cap^s T_i' = \mathbf{0}$
2. $T_i \cup^s T_i' = \mathbf{1}$

Figure 13 presents the complement T_1' of tree T_1 (see Figure 8 for T_1 and Figure 7 for T_G). Theorem 4.4 provides the

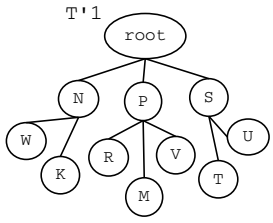

Figure 13. T_1': the complement of T_1.

support for complements.

Theorem 4.4 *Let* T_i, $\langle \{r\} \cup N_i, \mathcal{P}_i \rangle$ *be a tree in* $\mathcal{S}_G$, $\mathbf{0} \equiv T_0$ *and* $\mathbf{1} \equiv T_G$. *The tree* T_i', $\langle \{r\} \cup N_i', \mathcal{P}'_i \rangle$, *constructed as follows, is the complement of* T_i*:*

- $N_i' = N_G - N_i$*:*
  T_i' *has all nodes present in* T_G *and not in* T_i,

- $x\mathcal{P}'_i y$, *if* $x\mathcal{P}_G y$*:*
  T'_i *keeps all parent structural relationships* $p(x, y)$ *from* T_G *that involve nodes* x, y *present in* T_G *but not in* T_i *(*x, y *in* N'_i*).*

- $x\mathcal{P}'_i y$, *if* $\neg x\mathcal{P}_G y$ *and* $x\mathcal{P}^{tr}_G y$ *and all* $c_1, c_2, \ldots, c_{n-1}$ *in* $x\mathcal{P}_G c_1$, $c_1\mathcal{P}_G c_2$, $\ldots$, $c_{n-1}\mathcal{P}_G y$, $n \geq 2$, *are not in* N'_i*:*
  T *uses as parent relationships all ancestor relationships* $a(x, y)$ *from* T_G *that involve nodes* x, y *present in* T_G *but not in* T_i *(*$x, y \in N'_i$*), in the path of which all nodes do not belong to the set nodes present in* T_G *but not in* T_i*.*

Proof: $T'_i \subseteq^s T_G$, thus $T'_i \in \mathcal{S}_G$. The root node r is the only common node of T_i and T'_i, thus $T_0 \equiv \mathbf{0} = T_i \cap^s T'_i$. $(T_i \cup^s T'_i) \subseteq^s T_G$ and $(T_i \cup^s T'_i)$ has the same number of nodes with T_G, thus $T_i \cup^s T'_i = T_G \equiv \mathbf{1}$ (see Lemma 3.1). Finally, since $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a distributive lattice, the complement found for every tree is unique [7].

□

Since $\langle \mathcal{S}_G, \subseteq^s \rangle$

1. is a distributive lattice,
2. has $\mathbf{0}$ and $\mathbf{1}$, and
3. each T_i in $\mathcal{S}_G$ has a unique complement T_i in $\mathcal{S}_G$,

it is also a *boolean lattice* [7], getting all properties of boolean algebra.

Theorem 4.5 *The* $\mathcal{S}_G$ *equipped with the relation* $\subseteq^s$, $\langle \mathcal{S}_G, \subseteq^s \rangle$ *is a boolean lattice.*

Proof: Directly from Theorems 4.3 and 4.4

□

Exploiting the complements, we next define the last binary operator for structural reasoning on trees representing hierarchies for portal catalogs: S*-difference*.

4.2. S-difference ($-^s$)

Intuitively, the S*-difference* of two trees T_j and T_i, $T_j -^s T_i$, is the tree which provides structural information present in T_j and not in T_i. Figure 14 presents an example of an S-difference operation resulting to the tree $T_3 = T_2 -^s T_1$. The definition for the S-difference operation follows.

Definition 4.1 *Let* T_i, $\langle \{r\} \cup N_i, \mathcal{P}_i \rangle$, *and* T_j, $\langle \{r\} \cup N_j, \mathcal{P}_j \rangle$, *be two trees in* $\mathcal{S}_G$*. The* S-difference *of* T_j *and* T_i, $T_j -^s T_i$, *is the tree* $T = T_j \cap^s T'_i$.

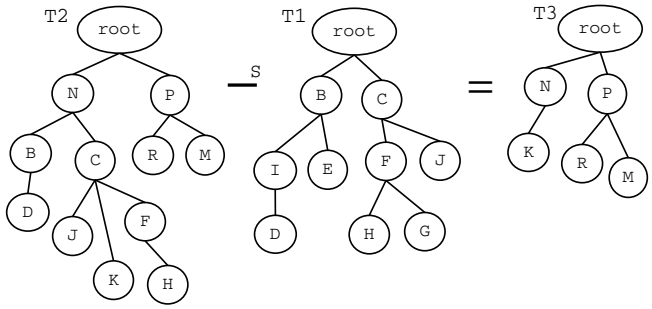

Figure 14. Difference operation: $T_3 = T_2 -^s T_1$.

Since $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a boolean lattice, the following laws involving the S-difference ($-^s$), the S-intersection $\cap^s$ and the S-union $\cup^s$ operators hold [7]:

- De Morgan's laws
  1. $T_i -^s (T_j \cup^s T_k) = (T_i -^s T_j) \cap^s (T_i -^s T_k)$
  2. $T_i -^s (T_j \cap^s T_k) = (T_i -^s T_j) \cup^s (T_i -^s T_k)$

5. Further discussion

In the previous Sections, we explored the algebraic properties of trees representing hierarchies of portal catalogs, based on the assumption that there is a global catalog available for the specific application domain or community. On the absence of such a global catalog, we need to provide a framework to ensure that similar algebraic properties hold. Our target is to construct such a global catalog, whenever possible, based on the available trees. Such a construction is based on the σ-union operator, which is similar to the S-union operator defined in previous Sections, but without using the global catalog. Similarly to S-union, the σ*-union* of two trees T_1 and T_2 ($T_1 \cup^\sigma T_2$) is the tree which provides integrated structural information from T_1 and T_2. However, its construction is based on the structural relationships coming from the involved trees.

Definition 5.1 *Let* T_i, $\langle \{r\} \cup N_i, \mathcal{P}_i \rangle$, *and* T_j, $\langle \{r\} \cup N_j, \mathcal{P}_j \rangle$, *be two trees in* $\mathcal{S} = \{T_1, T_2, \ldots\}$*. The* σ-union *of* T_i *and* T_j, $T_i \cup^\sigma T_j$, *is the tree* T, $\langle \{r\} \cup N, \mathcal{P} \rangle$, *constructed as follows:*

- $N = N_i \cup N_j$*:*
  T *has all nodes from both* T_i *and* T_j*.*

- $x\mathcal{P}y$, *if* $x\mathcal{P}_i y$ *and* $x\mathcal{P}_j y$*:*
  T *keeps all parent relationships* $p(x, y)$ *common to both* T_i *and* T_j *(*$x, y \in N$*).*

- $x\mathcal{P}y$, *if* $x\mathcal{P}_i y$ *and* $\neg x\mathcal{P}^{tr}_j y$, *or if* $x\mathcal{P}_j y$ *and* $\neg x\mathcal{P}^{tr}_i y$*:*
  T *keeps all parent relationships* $p(x, y)$ *from* T_i *(*T_j*) that involve nodes* x, y *related neither with parent*

$p(x, y)$ nor with ancestor relationship $a(x, y)$ in T_j (T_i).

The σ-union operation is vague under certain occasions. Consider the trees T_2 and T_5 in Figure 15. Which node, E or F, should be the parent of node C in the result? See also the trees T_2 and T_3 in Figure 15, where E is the parent of C in T_1, while E, C are children of the same node in T_2. Should the S-union keep the E, C as children or not?

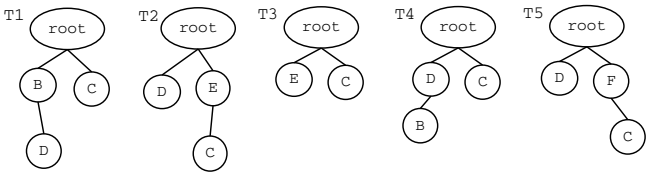

Figure 15.

We introduce the notions of *consistency* and *compatibility* for trees ([24]). Consistency ensures that the involved trees will not include nodes whose structural relationships are in conflict. For example, trees T_2 and T_3 in Figure 15 are not consistent, since E is the parent of C in T_1, while E, C are children of the same node in T_2. Compatibility ensures that we will always be able to decide the structural relationship between the involved nodes in the new tree produced from an operation like σ-union. For example, trees T_2 and T_5 in Figure 15 are not compatible, since it is not clear which node, E or F, should be the parent of node C in the result of an σ-union operation. Defining consistency and compatibility for trees, we suggest a valid σ-union operation on a set of trees [24], the output of which produces the global catalog T_G for the set of trees explored. Having such a tree T_G, we exploit the framework presented in the previous Sections to define the S-union, S-intersection and S-difference operators.

6. Application examples

Based on the motivating example in Section 1, we now express the queries mentioned there (see Figures 1, 2, 4, 5, 6) using the suggested S-union, S-intersection and S-difference operators. For the presented examples, T_G is the tree presented in Figure 3.

1. `Merge Adorama's and B&H catalogs`: $H_1 \cup^s H_2$.
2. `Find the common part of Adorama's and B&H catalogs`: $H_1 \cap^s H_2$.
3. `Find the part of Adorama's catalog which is not present in B&H's catalog`: $H_1 -^s H_2$.
4. `Merge RitzCamera's catalog with the common part of both Adorama's and B&H catalogs`: $H_3 \cup^s (H_1 \cap^s H_2)$.

Two more complicated examples follow:

1. `Find the part of Adorama's catalog which is not present in the merged catalog produced from B&H's and RitzCameras catalogs`: $H_1 -^s (H_2 \cup^s H_3)$. See the result in Figure 16.

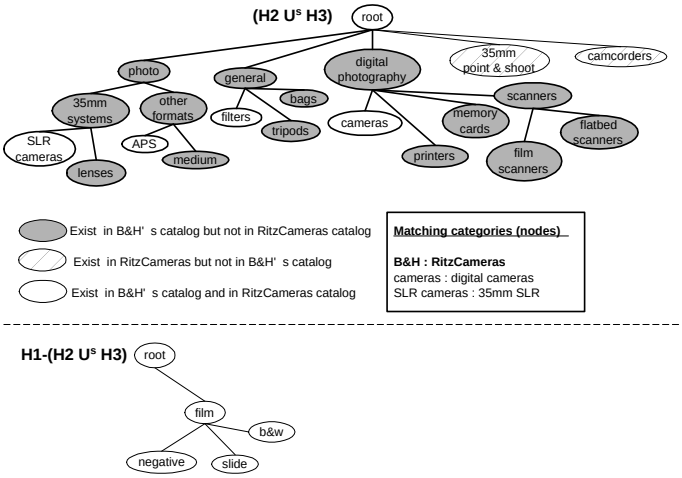

Figure 16.

2. `Take the part of Adorama's catalog which is not present in B&H's catalog and the part of Adorama's catalog which is not present in RitzCameras catalog, and find their common part`: $(H_1 -^s H_2) \cap^s (H_1 -^s H_3)$. See the result in Figure 17.

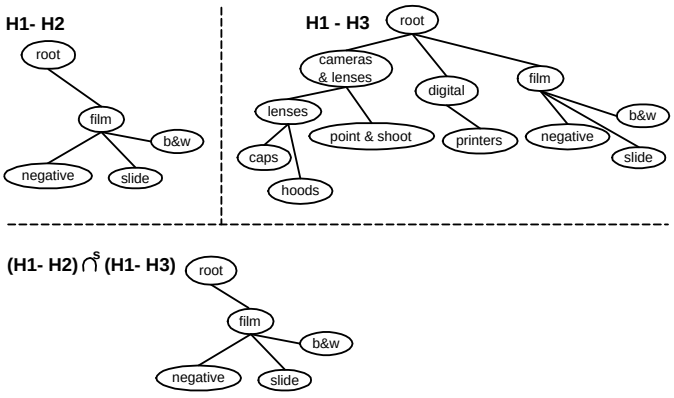

Figure 17.

Notice that the result of both queries is the same, since they obey one of the De Morgan's laws presented in the previous Section.

7. Conclusions and further work

With the spread of the use of portal catalogs and the XML infrastructure, there is a need to complement the existing portal catalog's querying capabilities with operations that manipulate structural information. This work suggested a framework for the structural manipulation of vertical portal catalog sites, that is catalogs of a specific subject or domain. We summarize our work as follows:

1. We presented a tree-like representation for hierarchies of vertical portal catalog sites.
2. We explored the algebraic properties of such trees and defined a lattice algebraic structure on them. Exploiting the upper and lower bounds of such a lattice, we formally defined the first two operators for structural manipulation of vertical portal catalog sites: S-union and S-intersection. The S-union operator provides an integrated form of all structural information from the involved trees. The S-intersection operator provides the common structural information from the involved trees.
3. Then, turning this lattice structure into a boolean algebra, we defined the S-difference operator based on the notion of complement trees. The S-difference operator provides structural information present in one tree and not in the other.
4. We gave the laws that hold for S-union, S-intersection and S-difference operators, so that one can transform and simplify sequences of operations, and, thus, optimize queries on structured data.
5. Finally, we identified the conditions under which this framework is applicable, using the notion of global catalog, which is either provided for the specific domain or constructed using the involved trees.

Our future work is based on two directions. First, we will further explore the algebraic properties of trees representing hierarchies of portal catalogs on the absence of a global catalog. Finally, we plan to integrate all suggested operators within a path-expression data query language to support both structural manipulation and data querying.

References

[1] Serge Abiteboul and Catriel Beeri, *The power of languages for the manipulation of complex values*, VLDB Journal **4** (1995), 727–793.

[2] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. Patel, D. Srivastava, and Y. Wu, *Structural joins: a primitive for efficient xml query processing*, Proceedings of ICDE Conference, 2002.

[3] F. Bancilhon and S. Khoshafian, *A calculus for complex objects*, Proceedings of ACM PODS Symposium on Principles of Database Systems (New York, USA), 1986, pp. 53–60.

[4] R. Behrens, *A grammar based model for XML schema integration*, Lecture Notes in Computer Science **1832** (2000).

[5] P. Buneman, S. Davidson, and A. Kosky, *Theoretical aspects of schema merging*, Proceedings of EDBT Conference (Berlin, Germany) (Alain Pirotte, Claude Delobel, and Georg Gottlob, eds.), March 1992, pp. 152–167.

[6] S. Y. Chien, V. J. Tsotras, C. Zaniolo, and D. Zhang, *Efficient complex query support for multiversion XML documents*, Proceedings of EDBT conference, 2002, pp. 161–178.

[7] B.A. Davey and H.A. Priestley, *Introduction to lattices and order*, Cambridge University Press, 2002.

[8] M. García-Solaco, F. Saltor, and M. Castellanos, *A structure based schema integration methodology*, Proceedings of ICDE Conference (Los Alamitos, CA, USA), March 1995, pp. 505–512.

[9] Robert C. Goldstein and Veda C. Storey, *Data abstractions: why and how?*, Data and Knowledge Engineering **29** (1999), 293–311.

[10] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu, *Approximate XML joins*, ACM SIGMOD Conference, 2002.

[11] Richard Hull and Roger King, *Semantic database modeling: survey, applications and research issues*, ACM Computing Surveys **19** (1987), no. 3, 201–260.

[12] P. Johannesson, *Using conceptual graph theory to support schema integration*, Lecture Notes in Computer Science **823** (1994).

[13] Y. Kalfoglou and M. Schorlemmer, *Information-flow-based ontology mapping*, Lecture Notes in Computer Science **2519** (2002).

[14] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and Michel Scholl, *Rql: A declarative query language for rdf*, Proceedings of WWW'02, Honolulu, Hawaii, USA, 2002.

[15] V. Kashyap and A. Sheth, *Semantic and schematic similarities between database objects: a context-based approach*, VLDB Journal **5** (1996), 276–324.

[16] W. Kent, *Solving domain mismatch and schema mismatch problems with an object-oriented database programming language*, Proceedings of VLDB Conference, Barcelona, Catalonia, Spain, 1991, pp. 147–160.

[17] W. Kim, H. T. Chou, and J. Banerjee, *Operations and implementation of complex objects*, Proceedings of ICDE Conference (Los Angeles, CA), February 1987.

[18] J.-L. Koh and A. L. P. Chen, *Integration of heterogeneous object schemas*, Lecture Notes in Computer Science **823** (1994).

[19] D. Lee and W. W. Chu, *Comparative analysis of six xml schema languages*, ACM SIGMOD Record **29** (2002), no. 3.

[20] D. Lee, M. Mani, and M. Murata, *"Reasoning about XML Schema Languages using Formal Language Theory"*, Tech. report, IBM Almaden Research, RJ#10197, Log#95071, Nov, 2000.

[21] J. Madhavan, P. A. Bernstein, and E. Rahm, *Generic schema matching with cupid*, Proceedings of VLDB Conference, Rome, Italy, 2001, pp. 49–58.

[22] V. M. Markowitz, *A relation merging technique for relational databases*, Proceedings of ICDE Conference (Los Alamitos, Ca., USA), IEEE Computer Society Press, February 1992, pp. 428–437.

[23] P. McBrien and A. Poulovassilis, *A formal framework for ER schema transformation*, Lecture Notes in Computer Science **1331** (1997), 408.

[24] Alexandra Meliou, *Modeling and exploring the algebraic properties of hierarchical structures*, KDBS Lab, Dept of Electrical and Computer Eng, NTU Athens, Jun 2003, Diploma Thesis (in Greek).

[25] R. S. Mello, S. Castano, and C. A. Heuser, *A method for the unification of xml schemata*, Information and Software Technology 44 (2002), 241–249.

[26] R. J. Miller, L. M. Haas, and M. A. Hernández, *Schema mapping as query discovery*, Proceedings of VLDB Conference, Cairo, Egypt, 2000, pp. 77–88.

[27] R. J. Miller, Y. Ioannidis, and R. Ramakrishnan, *Schema equivalence in heterogeneous systems: Bridging theory and practice*, Information Systems **19** (1994), no. 1, 3–31.

[28] T. Milo and S. Zohar, *Using schema matching to simplify heterogeneous data translation*, Proceedings of VLDB Conference, New York, NY, USA, 1998, pp. 122–133.

[29] B. Mitschang, *Extending the relational algebra to capture complex objects*, Proceedings of VLDB Conference, Amsterdam, The Netherlands (P. M. G. (Petrus Maria Gerardus) Apers and Gio Wiederhold, eds.), 1989, pp. 297–305.

[30] E. Rahm and P. A. Bernstein, *A survey of approaches to automatic schema matching*, VLDB Journal **10** (2001), no. 4, 334–350.

[31] E. A. Rundensteiner and L. Bic, *Set operations in a data model supporting complex objects*, Proceedings of EDBT Conference, 1990, pp. 286–300.

[32] I. Schmitt and G. Saake, *Merging Inheritance Hierarchies for Database Integration*, Proceedings of COOPS Conference, New York, USA, 1998.

[33] M. Siegel and S. E. Madnick, *A metadata approach to resolving semantic conflicts*, Proceedings of VLDB Conference, Barcelona (Catalonia, Spain) (Rafael Camps, Guy M. Lohman, and Amílcar Sernadas, eds.), 1991, pp. 133–145.

[34] W. W. Song, P. Johannesson, and J. A. Bubenko, *Semantic similarity relations in schema integration*, Lecture Notes in Computer Science **645** (1992).

[35] G. Stumme and A. Maedche, *FCA-MERGE: Bottom-up merging of ontologies*, Proceedings of IJCAI Conference, 2001, pp. 225–234.

[36] G. Wiederhold, *An algebra for ontology composition*, Proceedings of Monterey Workshop on Formal Methods, Monterey CA, 1994, pp. 56–61.