Modeling and Manipulating the Structure of Hierarchical Schemas for the Web

Theodore Dalamagas[1], Alexandra Meliou[2], Timos Sellis[1]

[1]School of Electr. and Comp. Engineering
National Techn. University of Athens, Hellas
{dalamag,timos}@dblab.ece.ntua.gr

[2]Department of Computer Science
University of California, Berkeley, USA
ameli@cs.berkeley.edu

Abstract

The Semantic Web is the next step of the current Web where information will become more machine-understandable to support effective data discovery and integration. Hierarchical schemas, either in the form of tree-like structures (e.g., DTDs, XML schemas), or in the form of hierarchies on a category/subcategory basis (e.g., thematic hierarchies of portal catalogs), play an important role in this task. They are used to enrich semantically the available information. Up to now, hierarchical schemas have been treated rather as sets of individual elements, acting as semantic guides for browsing or querying data. Under that view, queries like "find the part of a portal catalog which is not present in another catalog" can be answered only in a procedural way, specifying which nodes to select and how to get them. For this reason, we argue that hierarchical schemas should be treated as full-fledged objects so as to allow for their manipulation. This work proposes models and operators to manipulate the structural information of hierarchies, considering them as first-class citizens. First, we explore the algebraic properties of trees representing hierarchies, 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 hierarchies. These operators have certain algebraic properties to provide clear semantics and assist the transformation, simplification and optimization of sequences of operations using laws similar to those of set theory. Also, we identify the conditions under which this framework is applicable. Finally, we demonstrate an application of our framework for manipulating hierarchical schemas on tree-like hierarchies encoded as RDF/s files.

1 Introduction

The Internet is today's greatest source of information. Huge volumes of data is posted and retrieved through the Web. Despite this vast exchange of information, there is no consistent and strict organization of data. This, raises difficulties for data exchange and processing. For the Web to reach its full potential and become a universally accessible platform, the information should have well-defined meaning.

Hierarchical schemas play an important role in this task. They are used to enrich semantically the available information. Examples of such schemas include tree-like structures (e.g., DTDs, XML schemas), hierarchies on a category/subcategory basis (e.g., thematic hierarchies of portal

catalogs), etc. The importance of hierarchical schemas is even more evident in the context of Semantic Web. The Semantic Web is the next step of the current Web, where information should become more machine-understandable to support effective data discovery, automation and integration.

Up to now, hierarchical schemas (or simply hierarchies from now on) have been treated rather as sets of individual elements that provide semantic guidance to the users during browsing or querying data. However, in the Web environment, searching in a knowledge domain usually requires information processing in more than one sources related to that domain. These sources may employ different hierarchies to organize their data. New query requirements appear in this context. For instance, a user may need to "find the part of a portal catalog which is not present in another catalog". Under the traditional view of hierarchies as a set of individual nodes, a query like the one above can be answered only in a procedural way. That is, the user should explicitly specify which nodes to select and how to retrieve them. For this reason, we argue that hierarchical schemas should be treated as full-fledged objects, allowing for their manipulation as a whole. The next section clarifies such a motivation.

1.1 Motivating Example

Adorama, B&H and RitzCamera[1] are three e-market catalogs for photo equipment. Figure 1 shows parts of their hierarchy. Notice that a schema matching pre-processing identifies all matching categories (nodes) from both catalogs. Where matching is not straightforward due to different naming, we provide the necessary information giving the matching categories in Figures 1, 2 and 3.

One can think of various query operations on data provided by those catalogs. For example, browsing the hierarchies to find 35mm SLR cameras in all catalogs or posing path expression queries like /Filters/UV/"price<40", that is find ultra-violet filters with price less than 40euros. However, looking at the three e-marketplaces as *a set of similar hierarchies* with resources relevant to photo equipment, there is a need to complement such kind of querying with operations that manipulate structural information from hierarchies. Some examples follow:

(Q_1) Find the integrated hierarchy provided by Adorama's and B&H' hierarchies: Such a query has a 'union' flavor. Its answer should include structural information present either in Adorama or in B&H (i.e., merging Adorama with B&H).

(Q_2) Find the common part of Adorama's and B&H's hierarchies: Such a query has an 'intersection' flavor. Its answer should include structural information present in both Adorama and B&H.

(Q_3) Find the part of Adorama's hierarchy which is not present in B&H's hierarchy: Such a query has a 'difference' flavor. Its answer should include structural information present in Adorama but not in B&H.

(Q_4) Find the integrated structural information provided by (a) the common part of Adorama's and B&H hierarchies, and (b) RitzCamera's hierarchy.

Under the traditional view of hierarchies as a set of individual nodes, queries like Q_1, Q_2, Q_3, Q_4 can be answered only in a procedural way. That is, the user should specify which nodes to select

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

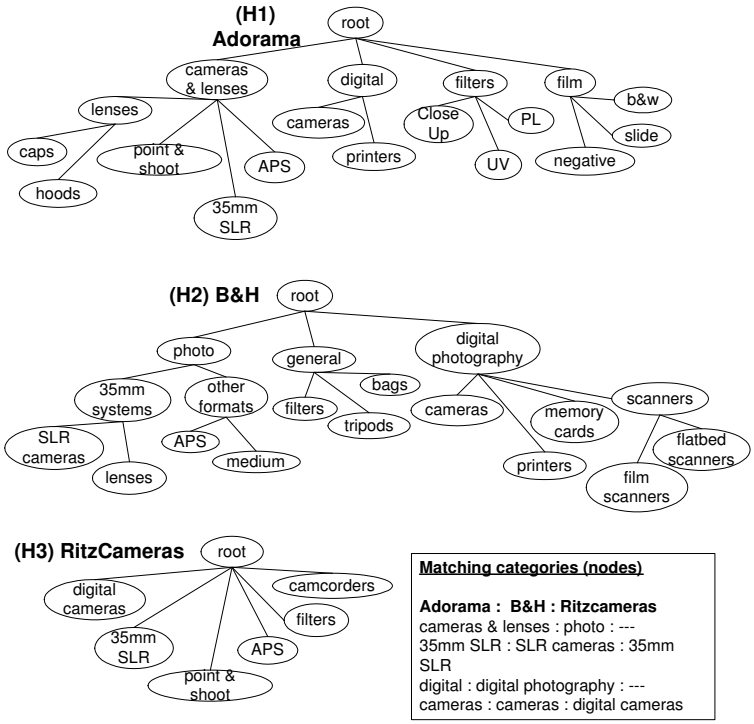

Figure 1: Part of Adorama's, B&H's and RitzCamera's hierarchy.

and how to retrieve them. In query Q_2 for instance, she should explicitly retrieve all common nodes and then construct a new hierarchy in such a way that the structural relationships are preserved. For this reason, we argue that answering such queries requires treating hierarchies as entities rather than set of individual nodes, and introducing a set of operators applied on hierarchies as a whole. Looking back at the first three example queries Q_1, Q_2, Q_3, we can identify three operators with union, intersection and difference semantics, respectively:

- Figure 2 presents the answer to the union query Q_1: a merged hierarchy from Adorama's and B&H hierarchies. In this new hierarchy, one can find all categories from Adorama's and B&H's. In Figure 3, the hierarchy shown is produced by merging the hierarchy in Figure 2 with RitzCameras' hierarchy.

- Figure 4 presents the answer to the intersection query Q_2: the common part of Adorama's and B&H hierarchies. In this new hierarchy, one can find categories common in Adorama and B&H. For example, the new hierarchy has APS and lenses for photo, a categorization which is applicable in both Adorama's (just below cameras & lenses, the matching category of photo) and B&H's hierarchy (following the path from photo down to the leaves).

- Figure 5 presents the part of Adorama's hierarchy not present in B&H's hierarchy, which is the answer to the difference query Q_3. For example, the new hierarchy has negative, slide and b&w for film, a categorization found only in Adorama.

- Finally, Figure 6 shows the answer of Q_4, which is a more complex query. Notice that Q_4 is actually an intersection query, followed by a union query.

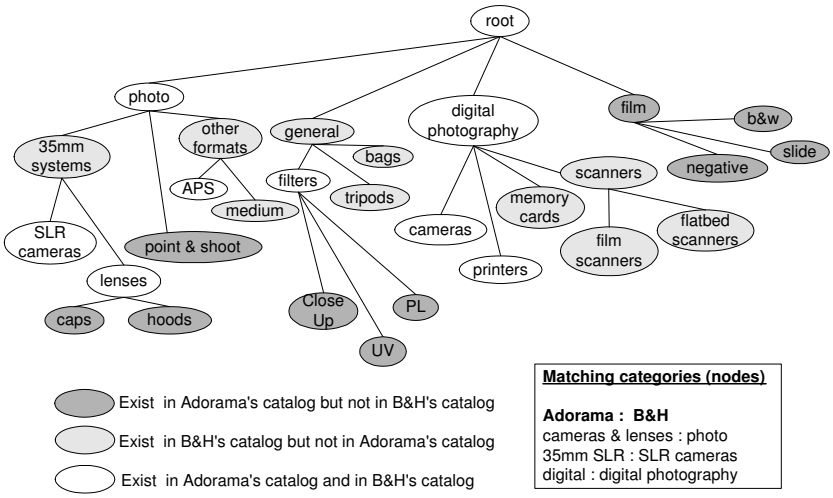

Figure 2: Merging Adorama's and B&H's hierarchies.

We consider such kind of query requirements as part of the *generic model management* framework, presented in [20]. According to this framework, models are manipulated as abstractions rather than sets of individual elements, using *model-at-a-time* and *mapping-at-a-time* operators. However, these operators are rather high-level operators without certain algebraic properties. There are several cases where such properties are required:

1. When applying an operator on a set of hierarchies, the result should be the same regardless the sequence of the operations.

2. The required operators have union, intersection and difference semantics. Given that laws similar to those of set theory hold, one should be able to transform, simplify and optimize sequences of operations. For example, given the merged hierarchy of RitzCameras and Adorama, and the merged hierarchy of RitzCameras and B&H's, one could answer query Q_4 by just finding their common part, given that a kind of a distributive law holds.

3. Finally, there are several cases where we need to check whether these operations are applicable in the presence of structural inconsistencies. Having for example the hierarchy H_1 in Figure 1 and a similar hierarchy, but with camera to be the parent of digital, we cannot get a merged hierarchy unless we decide what will be the new relationship between camera and digital.

In our case, we aim to manipulate hierarchies with low-level, set-like query operators applied on trees. We emphasize on studying their algebraic properties and providing clear semantics. These properties can assist the transformation, simplification and optimization of sequences of operations on hierarchies.

1.2 Contribution

The main contribution of this work is a set of low-level, query operators with set-like semantics applied on hierarchical schemas as full-fledged objects. Specifically:

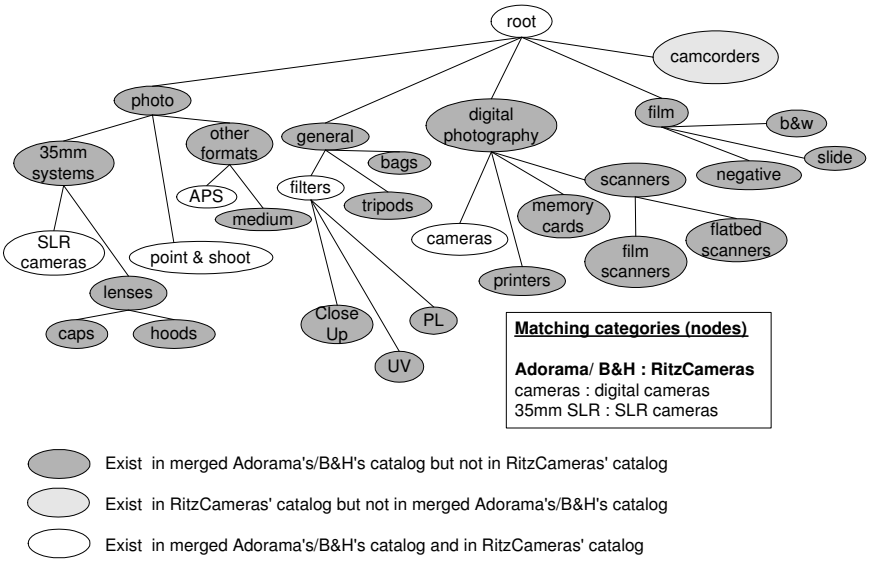

Figure 3: Merging Adorama/B&H's and RitzCameras' hierarchies.

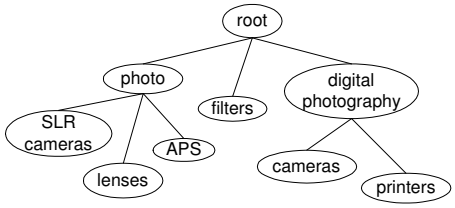

Figure 4: The common part of Adorama's and B&H's hierarchies.

1. We study the algebraic properties of trees representing hierarchies, based on the assumption that a global tree is given. We introduce the S-subset tree relation, and we define a partial order for hierarchies based on that relation.

2. We define the S-union and S-intersection operators to manipulate the structure of hierarchies. Both operators are binary. Given two hierarchies, the S-union operator produces the merged hierarchy. The S-intersection operator extracts the common part of two hierarchies.

3. Then, we prove that the S-union and the S-intersection operators are actually the least upper bound and the greatest lower bound of the trees involved, respectively. Using this result, we turn the proposed partial order into a lattice.

4. We define the notion of complement trees, and we turn the lattice into a boolean lattice, getting all properties of a boolean algebra. Also, using complements, we define the S-difference operator. Given two hierarchies, the S-difference operator extracts the part of the first hierarchy which is not present in the second hierarchy.

5. We give the laws that hold for these operators so that one can transform, simplify and optimize sequences of operators. The laws are similar to those in set theory.

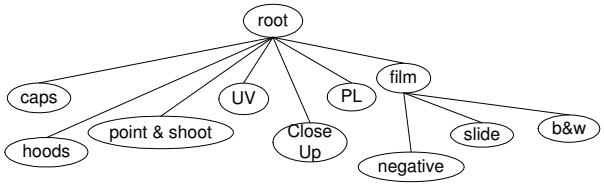

Figure 5: The part of Adorama's hierarchy which is not present in B&H's hierarchy.

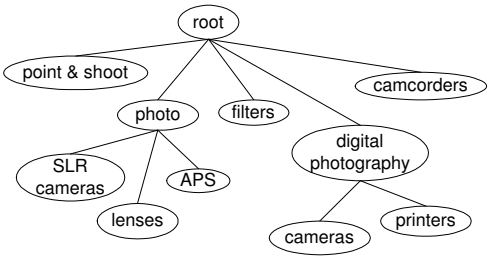

Figure 6: Merging RitzCamera's hierarchy with the common part of both Adorama's and B&H hierarchy.

6. We define a framework to ensure that similar algebraic properties and laws hold, in the absence of a global tree. We present the conditions under which such a tree can be constructed using the available trees.

7. We demonstrate our framework for manipulating hierarchical schemas on tree-like hierarchies encoded as RDF/s files, using a prototype system that implements the S-union, S-intersection and S-difference operators.

1.3 Related Work

Related issues have been considered 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 [16, 13, 18, 11, 17, 25, 12, 31]. Theoretical aspects of schema merging have been discussed in [6, 21]. Finding similarities between objects of different schemas and dealing with semantic conflicts to support schema matching is crucial for schema merging and information integration. For example, [10] introduces a method to detect semantically related classes in object-oriented multidatabase systems. For a detailed survey on schema matching issues one can see [26, 9]. Most recent research focuses on (a) ontologies [28, 27, 12, 5, 29], where methods for ontology composition and merging are presented (see [23] for a survey), and (b) XML schemas trees [19, 4, 7], where methods to unify and integrate trees representing XML data are discussed. Also, in [30], the authors present schema matching techniques for XML data using a similarity measure and relaxation labeling. However, the presented related work is 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. Integration of schemas is considered as the task which produces a global schema

to cover all those schemas. Our work, on the other hand, supports the construction of integrated schemas (hierarchies), using operations with union, intersection and difference semantics. These operations are applied on schemas as a whole, and not as a set of individual elements.

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. A calculus for complex objects is presented in [3]. A lattice structure is defined on nested objects and operators on these objects are introduced. However, the operators 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 [15]. Again the target is 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. In [22], an extension of the relational algebra is suggested to capture complex objects. Other works, like [24], 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 not on supporting structural manipulation on structures as a whole.

1.4 Outline

Section 2 discusses modeling issues for hierarchies, studies the algebraic properties of trees representing hierarchies, and introduces the S-union and S-intersection operators. Section 3 defines a lattice algebraic structure on these trees, and turns this structure into a boolean algebra. Then, the section introduces the S-difference operator, and presents a set of laws hold. Section 4 presents a framework to ensure that similar algebraic properties and laws hold in the absence of a global tree. Section 5 demonstrates a case study for manipulating hierarchical schemas on tree-like hierarchies encoded as RDF/s files. Finally, Section 6 concludes this paper.

2 Modeling Issues and Algebraic Properties of Hierarchies

Data on the Web are *semistructured data* [2], in the sense that they are schemaless and self-describing pieces of information. Hierarchical schemas are used to semantically enrich data on the Web. Examples of such schemas are usually tree-like structures with syntactic constraints and type information (e.g., DTDs, XML schemas, etc), thematic hierarchies, etc. Portal catalogs are examples that exploit thematic hierarchies to organize data, and they will be used as test cases throughout this work.

Figure 7 shows the hierarchies S_1 and S_2 of two portal catalogs for photo equipment. *Global hierarchies* for certain domains usually help portal administrators to build their own (local) hierarchies. A local hierarchy is actually a part of a global hierarchy, having only the categories needed according to the application requirements. This prevents the existence of structural inconsistencies in local hierarchies. Figure 8 presents an example of a global hierarchy S for photo equipment. The hierarchies S_1 and S_2 (Figure 7) are parts of S. Simple name mismatches can be easily resolved using some schema matching tool (see related work). For example, Single Reflex and ultra violet in S_1 are matched to SLR and uv in S_2, respectively. On the other hand,

if such a global hierarchy is not available, structural inconsistencies may occur. See for example the catalog S_1 in Figure 7. The author of another catalog similar to S_1 might prefer having the categorization new and second hand as a top level choice during browsing.

Next, we study the algebraic properties of trees representing hierarchies. Initially, we assume that there is a global hierarchy available for the specific application domain or community. In Section 4, we relax this assumption.

2.1 Hierarchies as Tree-like Structures

We consider hierarchies as tree-like structures. Figures 7 and 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 hierarchy that match nodes of another hierarchy.

Definition 2.1. *A* tree*, T is (a) a* root *node r or (b) a* root *node r and a set $\mathcal{E}$ of edges, $\mathcal{E} = \{e_1, e_2, \ldots, e_k\}$, with $e_1 : r \rightarrow r_1$, $e_2 : r \rightarrow r_2$, $\ldots e_k : r \rightarrow r_k$, where $r_1, r_2, \ldots r_k$ are root nodes of k (sub)trees.*

A function $label(n)$ assigns a string label as the identifier for every node n of T. Especially for r, $label(r) =$root. The tree that has root r as its only node is denoted as T_0. Node A is the *parent* of node B, denoted as $p(A, B)$, if there is a direct edge from A to B. Node A is an *ancestor* of node B, denoted as $a(A, B)$, if there is a direct path of k edges, $k > 1$, from A to B. A tree T with nodes $N = \{n_1, n_2, \ldots, n_k\}$ and root r can be represented as a relation $\mathcal{P}$ on the set $\{r\} \cup N$, denoted as $\langle \{r\} \cup N, \mathcal{P} \rangle$: $x\mathcal{P}y$ holds if $p(x, y)$ holds, that is node x is the parent of node y, with $x, y \in \{r\} \cup N$.

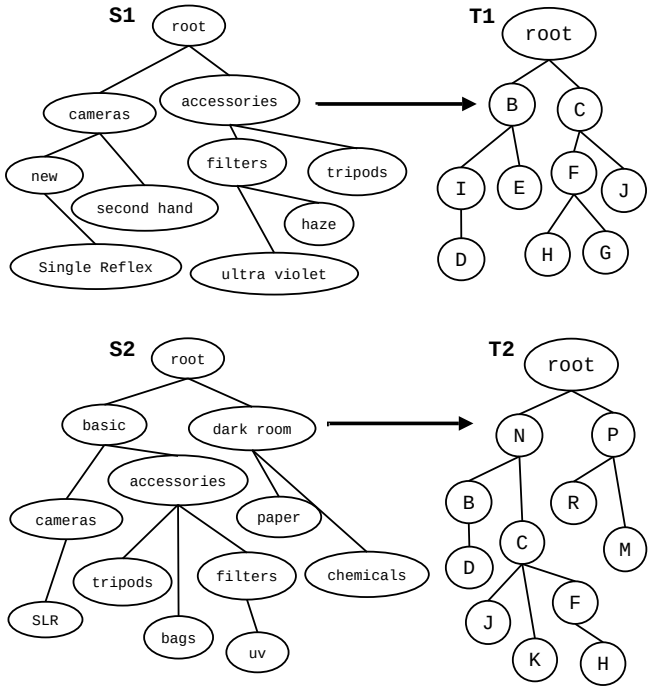

Figure 7: Catalogs S_1, S_2 and their representative trees T_1 and T_2.

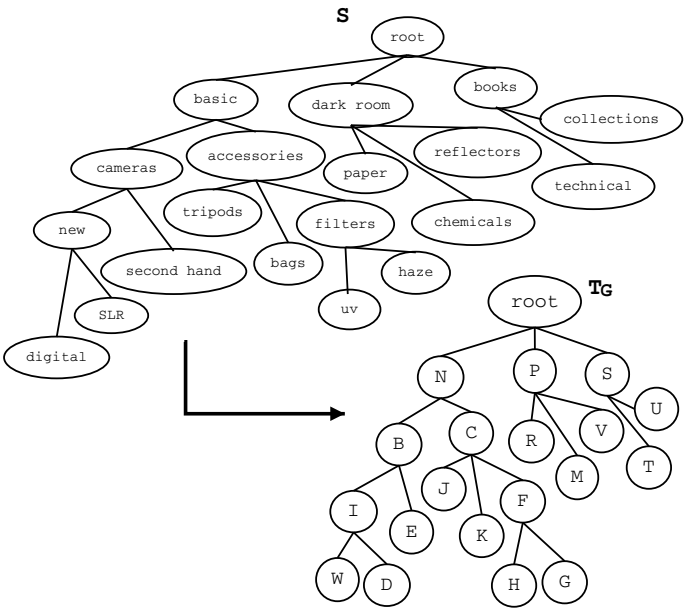

Figure 8: A global hierarchy S for photo equipment and its representative global tree T_G.

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

Definition 2.2. *$T_i = T_j$, iff $N_i = N_j$ and $\forall\, x, y \in \{r\} \cup N_i$ 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 $x, y \in \{r\} \cup N$ 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. Also, by deleting all nodes of T_2 not present in T_1 and moving their children in higher levels we will obtain T_1.

Definition 2.3. *A tree $T_i = \langle \{r\} \cup N_i, \mathcal{P}_i \rangle$ is an S-subset of a tree $T_j = \langle \{r\} \cup N_j, \mathcal{P}_j \rangle$, denoted as $T_i \subseteq^s T_j$, if $\forall\, x, y \in \{r\} \cup N_i$ with $x\mathcal{P}_i y$ then $x\mathcal{P}_j^{tr} y$ and if $\forall\, x, y \in \{r\} \cup N_i$ with $x\mathcal{P}_j^{tr} y$ then $x\mathcal{P}_i^{tr} y$. Especially for T_0, $T_0 \subseteq^s T$ for any $T = \langle \{r\} \cup N, \mathcal{P} \rangle$.*

For example, in Figure 9, $T_1 \subseteq^s T_3$. T_1 can be constructed by deleting nodes D and C from T_3 and moving E and F up in a higher level. $T_2 \not\subseteq^s T_3$ since $p(E, C)$ holds in T_2 but not in T_3. Also, $T_2 \not\subseteq^s T_4$ since $p(root, C)$ in T_4 but $a(root, C)$ in T_2. This latter example is a case where the second requirement of Definition 2.3 is neccessary. Note that by examining only the first requirement would give $T_4 \subseteq^s T_2$.

We next present two lemmas needed for subsequent theorems. According to the first one, two trees are equal if they have the same number of nodes and are related with an S-subset relation. According to the second one, the node set of a tree is a subset of the node set of another one if these two trees are related with an S-subset relation.

Lemma 2.1. *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 2.3 and 2.2. □

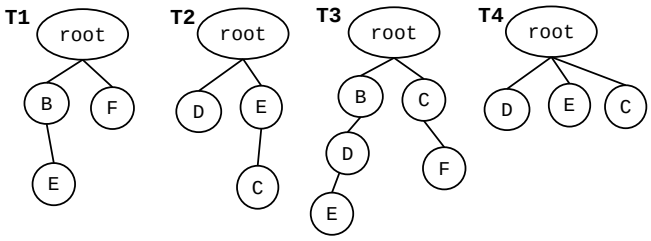

Figure 9: $T_1 \subseteq^s T_3$, $T_2 \not\subseteq^s T_3$.

Lemma 2.2. *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 where $T_i \subseteq^s T_j$. Then, $N_i \subseteq N_j$.*

Proof. Directly from Definition 2.3. □

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. Figure 8 shows an example of a global tree T_G and Figure 7 shows trees T_1 and T_2, with $T_1 \subseteq^s T_G$ and $T_2 \subseteq^s T_G$. Note that $\mathcal{S}_G$ includes T_0, which is the tree with only one node (its root), and T_G. Based on the following Theorem 2.1, 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 2.1. *The $\subseteq^s$ relation on $\mathcal{S}_G$, $\langle \mathcal{S}_G, \subseteq^s \rangle$, is reflexive ($T_i \subseteq^s T_i$), antisymmetric ($T_i \subseteq^s T_j$ and $T_j \subseteq^s T_i$ imply $T_i = T_j$), and transitive ($T_i \subseteq^s T_j$ and $T_j \subseteq^s T_k$ imply $T_i \subseteq^s T_k$).*

Proof. Reflexivity holds (directly from Definition 2.3). Antisymmetry 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 2.1). Transitivity holds, too. If $T_i \subseteq^s T_j$ and $T_j \subseteq^s T_k$, then

$$\forall\, x, y \in \{r\} \cup N_i \, with \, x \mathcal{P}_i y \Rightarrow x \mathcal{P}_j^{tr} y \tag{1}$$

$$\forall\, x, y \in \{r\} \cup N_i \, with \, x \mathcal{P}_j^{tr} y \Rightarrow x \mathcal{P}_i^{tr} y \tag{2}$$

$$\forall\, x, y \in \{r\} \cup N_j \, with \, x \mathcal{P}_j y \Rightarrow x \mathcal{P}_k^{tr} y \tag{3}$$

$$\forall\, x, y \in \{r\} \cup N_j \, with \, x \mathcal{P}_k^{tr} y \Rightarrow x \mathcal{P}_j^{tr} y \tag{4}$$

From (1), for all $x, y \in \{r\} \cup N_i$ with $x \mathcal{P}_i y$, there exist $c_0 = x, c_1, \ldots, c_n = y$ with $c_p \mathcal{P}_j c_{p+1}$ for all $0 \leq p < n$. For $n = 1$, $x \mathcal{P}_j y$ holds, and, thus, (3) 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 (from (3))

(for $x \mathcal{P}_j c_1$) $\exists\, c_0^1 = x, c_1^1, \ldots, c_{k1}^1 = c_1$ so that $x \mathcal{P}_k c_1^1, c_1^1 \mathcal{P}_k c_2^1 \ldots c_{k1-1}^1 \mathcal{P}_k c_1$, and
(for $c_1 \mathcal{P}_j c_2$) $\exists\, c_0^2 = c_1, c_1^2, \ldots, c_{k2}^2 = c_2$ so that $c_1 \mathcal{P}_k c_1^2, c_1^2 \mathcal{P}_k c_2^2, \ldots, c_{k2-1}^2 \mathcal{P}_k c_2$, and
$\ldots$
(for $c_{n-1} \mathcal{P}_j y$) $\exists\, c_0^n = c_{n-1}, c_1^n, \ldots, c_{kn}^n = y$ so that $c_{n-1} \mathcal{P}_k c_1^n, c_1^n \mathcal{P}_k c_2^n, \ldots, c_{kn-1}^n \mathcal{P}_k y$,

that is $x \mathcal{P}_k^{tr} y$ holds. So, in any case ($n = 1$ or $n > 1$): $\forall\, x, y \in \{r\} \cup N_i \, with \, x \mathcal{P}_i y, \Rightarrow x \mathcal{P}_k^{tr} y$, that is the first requirement of Definition 2.3 is fulfilled. We next explore the second requirement of Definition 2.3. Using Lemma 2.2, we get $N_i \subseteq N_j$, so (4) holds for all $x, y \in \{r\} \cup N_i$, too:

$\forall\, x, y \in \{r\} \cup N_i \, with \, x\mathcal{P}_k^{tr} y \Rightarrow x\mathcal{P}_j^{tr} y$, thus (using (2))
$\forall\, x, y \in \{r\} \cup N_i \, with \, x\mathcal{P}_k^{tr} y \Rightarrow x\mathcal{P}_i^{tr} y$,

that is the second requirement of Definition 2.3 is fulfilled, too. □

We note that the partial order $\langle \mathcal{S}_G, \subseteq^s \rangle$ has T_0 as its bottom element $\perp$, and T_G as its top element $\top$ ($\perp \subseteq^s T_i$ and $T_i \subseteq^s \top$ for all $T_i \in \mathcal{S}_G$). We next define the first two binary operators for structural manipulation of trees in $\mathcal{S}_G$: *S-union* and *S-intersection*.

2.2 *S*-union ($\cup^s$)

Intuitively, the *S-union* of two trees $T_i, T_j \in \mathcal{S}_G$, $T_i \cup^s T_j$, is the tree $T \in \mathcal{S}_G$ which provides integrated structural information from T_i and T_j. Figure 10 presents an example of an S-union operation on trees T_1 and T_2, both S-subsets of T_G (Figure 8), resulting to the tree $T_3 = T_1 \cup^s T_2$. T_3 contains all nodes from both trees T_1 and T_2. The structural relationships in T_3 can be determined as follows. In case of relationships of different type which involve the same pair of nodes in T_1 and T_2, only the ancestor ones are preserved in T_3. For example, since $a(B, D)$ exists in T_1 and $p(B, D)$ exists in T_2, T_3 keeps $a(B, D)$. All the other structural relationships in T_1 and T_2 are preserved in T_3.

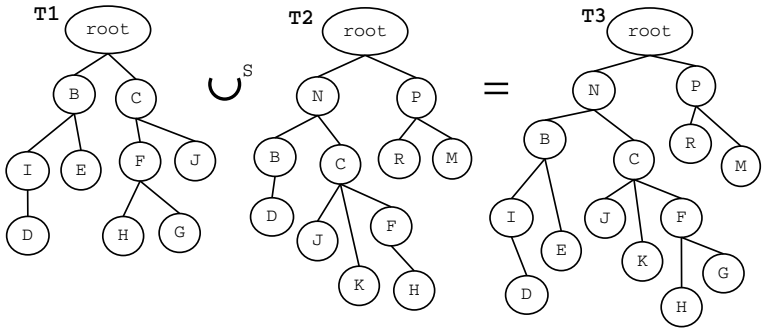

Figure 10: Union operation: $T_3 = T_1 \cup^s T_2$.

A more interesting case is presented in Figure 11, which shows a variation of trees T_1 and T_2 used in the previous example. Note that the new T_1 lacks node H, while T_2 lacks node F. T_3' is the tree produced by $T_1 \cup^s T_2$, following the same way to determine the structural relationships as in the previous example. However, since T_3' lacks $p(F, H)$ which is present in T_G, $T_3' \not\subseteq^s T_G$. Thus, $T_1 \cup^s T_2$ should also contain relationships neither present in T_1 nor in T_2, but imposed by T_G, like for example $p(F, H)$. Taking $p(F, H)$ into consideration, we get T_3 (Figure 11). The formal definition for the S-union operation follows.

Definition 2.4. *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* $x\mathcal{P}_G y$, *with* $x, y \in \{r\} \cup N$:
  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.

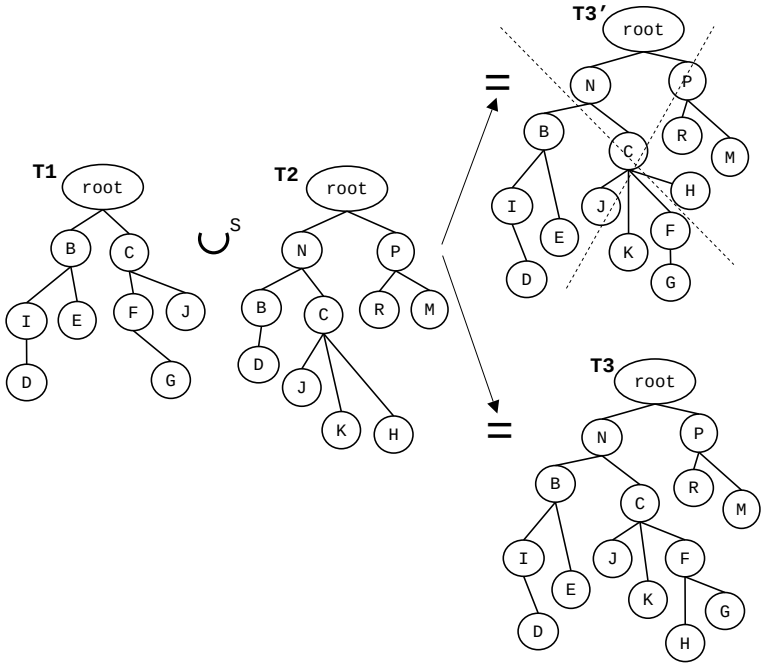

Figure 11: Union operation: $T_3 = T_1 \cup^s T_2$.

- *$x\mathcal{P}y$, 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 $(x, y \in \{r\} \cup N)$:*
  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, in the path of which all nodes belong neither to T_i nor to T_j.

We note that

1. $(T_i \cup^s T_j) \subseteq^s T_G$, 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. We can construct $T_1 \cup^s T_2$, with $T_1 = \langle \{r\} \cup N_1, \mathcal{P}_1 \rangle$ and $T_2 = \langle \{r\} \cup N_2, \mathcal{P}_2 \rangle$, by deleting all nodes of T_G not present in $N_1 \cup N_2$ and moving their children in higher levels accordingly. This task can be performed by traversing T_G, e.g., using depth-first search, and checking whether each visited node exists in $N_1 \cup N_2$. Assuming a hash-based check, the cost for the union operation is $O(|V| + |E|)$, where $|V|$ is the number of nodes and $|E|$ is the number of edges in T_G.

2.3 S-intersection ($\cap^s$)

Intuitively, the *S-intersection* of two trees $T_i, T_j \in \mathcal{S}_G$, $T_i \cap^s T_j$, is the tree $T \in \mathcal{S}_G$ which provides structural information common in T_i and T_j. Figure 12 presents an example of an S-intersection operation on trees T_1 and T_2, both S-subsets of T_G (Figure 8), resulting to the tree

$T_3 = T_1 \cap^s T_2$. T_3 contains all common nodes in trees T_1 and T_2. The structural relationships in T_3 can be determined as follows. In case of relationships of different type which involve the same pair of nodes in T_1 and T_2, only the parent ones are preserved in T_3. For example, since $a(B, D)$ in T_1 and $p(B, D)$ in T_2, T_3 keeps $p(B, D)$. An ancestor relation that involves the same pair (x, y) of nodes in T_1 and T_2 is preserved in T_3 only if the set of nodes after x and before y in T_1 is the same with the set of nodes after x and before y in T_2. Otherwise, it becomes a parent relation in T_3. For example, since $a(C, H)$ in T_1, $a(C, H)$ in T_2 and node F is a common node in T_1 and T_2, T_3 keeps $a(C, H)$. The definition for the S-intersection operation follows.

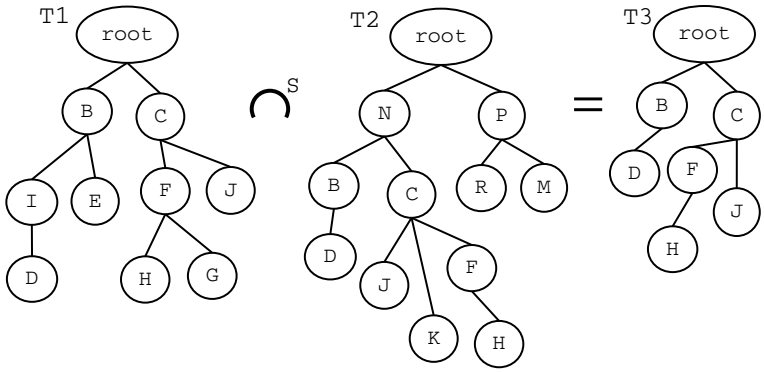

Figure 12: Intersection operation: $T_3 = T_1 \cap^s T_2$.

Definition 2.5. *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 $x\mathcal{P}_G y$, with $x, y \in \{r\} \cup N$:*
  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.
- *$x\mathcal{P}y$, 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*
  $x\mathcal{P}_G c_1$, $c_1\mathcal{P}_G c_2$, …, $c_{n-1}\mathcal{P}_G y$, $n \geq 2$, are not in N $(x, y \in \{r\} \cup N)$:
  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, 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. $(T_i \cap^s T_j) \subseteq^s T_G$, 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.
3. Similarly to S-union, we can construct $T_1 \cap^s T_2$, with $T_1 = \langle \{r\} \cup N_1, \mathcal{P}_1 \rangle$ and $T_2 = \langle \{r\} \cup N_2, \mathcal{P}_2 \rangle$, by deleting all nodes of T_G not present in $N_1 \cap N_2$ and moving their

children in higher levels accordingly.
The cost for the intersection operation is $O(|V| + |E|)$, where $|V|$ is the number of nodes and $|E|$ is the number of edges in T_G.

3 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 3.1 shows, $\cup^s$ gives the *least upper bound*, while $\cap^s$ gives the *greatest lower bound*.

Theorem 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 in $\mathcal{S}_G$. The set of all upper bounds of $\mathcal{S}_G$ is $\mathcal{S}_G^u = \{T_k \in \mathcal{S}_G : T_i \subseteq^s T_k \text{ and } T_j \subseteq^s T_k\}$. The set of all lower bounds of $\mathcal{S}_G$ is $\mathcal{S}_G^l = \{T_k \in \mathcal{S}_G : T_k \subseteq^s T_i \text{ and } T_k \subseteq^s T_j\}$. 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 $T_j \subseteq^s T$. The node set N of T is a subset of the node set N' of $T_i \cup^s T_j$: $N \subseteq N'$ (Lemma 2.2). If $N \subset N'$, neither $T_i \subseteq^s T$ nor $T_j \subseteq^s T$ can hold. If $N = N'$, $T = T_i \cup^s T_j$ (Lemma 2.1). So there is not such a tree T other than $T_i \cup^s T_j$. Similarly, we can show that there is not a tree T for which $(T_i \cap^s T_j) \subseteq^s T$, $T \subseteq^s T_i$ and $T \subseteq^s T_j$. □

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 $\subseteq^s$, $\langle \mathcal{S}_G, \subseteq^s \rangle$. Thus, $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a lattice.

Theorem 3.2. *The $\mathcal{S}_G$ equipped with the relation $\subseteq^s$, $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a lattice.*

Proof. Directly from Theorems 2.1 and 3.1. □

Since $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a lattice, idempotency, commutative, associative and absorption laws hold [8], as Table 1 shows. We note (see [8]) 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,

$$a \wedge (b \vee c) \geq (a \wedge b) \vee (a \wedge c) \tag{5}$$

and

$$a \vee (b \wedge c) \geq (a \vee b) \wedge (a \vee c) \tag{6}$$

that is equality in the distributive laws for lattices does not hold in general. Theorem 3.3 shows that $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a distributive lattice, that is the distributive laws hold (see Table 1).

Theorem 3.3. *The $\mathcal{S}_G$ equipped with the relation $\subseteq^s$, $\langle \mathcal{S}_G, \subseteq^s \rangle$, is a distributive lattice.*

Idempotency laws	$T_i \cap^s T_i = T_i$ $T_i \cup^s T_i = T_i$
Commutative laws	$T_i \cap^s T_j = T_j \cap^s T_i$ $T_i \cup^s T_j = T_j \cup^s T_i$
Associative laws	$T_i \cap^s (T_j \cap^s T_k) = (T_i \cap^s T_j) \cap^s T_k$ $T_i \cap^s (T_j \cap^s T_k) = (T_i \cap^s T_j) \cap^s T_k$
Absorption laws	$T_i \cap^s (T_i \cup^s T_j) = T_i$ $T_i \cup^s (T_i \cap^s T_j) = T_i$
Distributive laws	$T_i \cup^s (T_j \cap^s T_k) = (T_i \cup^s T_j) \cap^s (T_i \cup^s T_k)$ $T_i \cap^s (T_j \cup^s T_k) = (T_i \cap^s T_j) \cup^s (T_i \cap^s T_k)$
De Morgan's laws	$T_i -^s (T_j \cup^s T_k) = (T_i -^s T_j) \cap^s (T_i -^s T_k)$ $T_i -^s (T_j \cap^s T_k) = (T_i -^s T_j) \cup^s (T_i -^s T_k)$

Table 1: Laws hold for $\langle \mathcal{S}_G, \subseteq^s \rangle$.

Proof. Let $T_i = \langle \{r\} \cup N_i, \mathcal{P}_i \rangle$, $T_j = \langle \{r\} \cup N_j, \mathcal{P}_j \rangle$, and $T_k = \langle \{r\} \cup N_k, \mathcal{P}_k \rangle$, all members of $\mathcal{S}_G$. Also, let $T_1 = T_i \cup^s (T_j \cap^s T_k) = \langle \{r\} \cup N_1, \mathcal{P}_1 \rangle$ and $T_2 = (T_i \cup^s T_j) \cap^s (T_i \cup^s T_k) = \langle \{r\} \cup N_2, \mathcal{P}_2 \rangle$. Using Definitions 2.4 and 2.5, $N_1 = N_i \cup (N_j \cap N_k)$ and $N_2 = (N_i \cup N_j) \cap (N_i \cup N_k)$, thus $N_1 = N_2$.

Recall that the output of an S-union (or S-intersection) operator is the tree constructed by deleting all nodes of T_G not present in the union of nodes in the involved trees (or in the intersection of nodes in the involved trees) and moving their children in higher levels accordingly. Thus, since T_1 and T_2 involve the same nodes from T_G, $T_1 = T_2$. □

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

3.1 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. Figure 13 presents the complement T_1' of tree $T_1 = \langle \{r\} \cup N_1, \mathcal{P}_1 \rangle$ (see Figure 7 for T_1 and Figure 8 for T_G). T_1' keeps the root of T_G and all of its nodes not present in T_1. We note that given T_i, T_i' in $\mathcal{S}_G$, $T_i \cap^s T_i' = \mathbf{0}$ and $T_i \cup^s T_i' = \mathbf{1}$, where

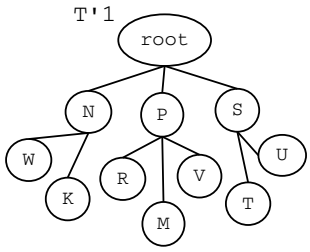

Figure 13: T_1': the complement of T_1.

$\mathbf{0}$ and $\mathbf{1}$ are two unique trees in $\mathcal{S}_G$.

Theorem 3.4. *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 $\mathbf{1} \equiv T_G = \langle \{r\} \cup N_G, \mathcal{P}_G \rangle$. The tree $T_i' = \langle \{r\} \cup N_i', \mathcal{P}'_i \rangle$, constructed as follows, is the unique* 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* $x\mathcal{P}_G y$*, with* $x, y \in \{r\} \cup N'_i$*:*
  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*.*

- $x\mathcal{P}'_i y$*, 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'_i *(*$x, y \in \{r\} \cup N'_i$*):*
  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*, 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 2.1). Finally, since $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a distributive lattice, the complement found for every tree is unique [8]. □

We can cosntruct T'_1 by deleting all nodes of T_G not present in $N_G - N_1$ and moving their children in higher levels accordingly. The cost for the operation is $O(|V| + |E|)$, where $|V|$ is the number of nodes and $|E|$ is the number of edges in T_G.

Since $\langle \mathcal{S}_G, \subseteq^s \rangle$ (a) is a distributive lattice, (b) has $\mathbf{0} \equiv T_0$ and $\mathbf{1} \equiv T_G$, and (c) each T_i in $\mathcal{S}_G$ has a unique complement T_i in $\mathcal{S}_G$, it is also a *boolean lattice* [8], getting all properties of a boolean algebra.

Theorem 3.5. *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 3.3 and 3.4. □

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

3.2 *S*-difference ($-^s$)

Intuitively, the *S-difference* of two trees $T_j, T_i \in \mathcal{S}_G$, $T_j - ^s T_i$, 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 $T_3 = T_2 -^s T_1$. T_3 can be constructed using complements: $T_3 = T_2 \cap^s T'_1$. The definition for the S-difference operation follows.

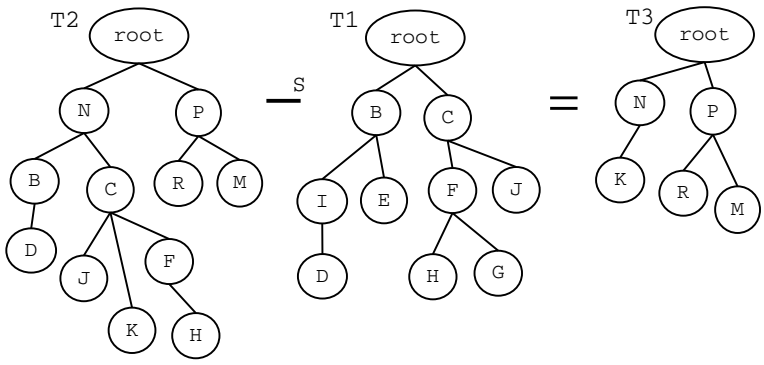

Figure 14: Difference operation: $T_3 = T_2 -^s T_1$.

Definition 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 in* $\mathcal{S}_G$*. The* S-difference *of* T_j *and* T_i*,* $T_j -^s T_i$*, is the tree* $T = T_j \cap^s T_i'$*.*

Since $\langle \mathcal{S}_G, \subseteq^s \rangle$ is a boolean lattice, the De Morgan's laws hold [8]. Table 1 shows all laws hold for $\langle \mathcal{S}_G, \subseteq^s \rangle$.

3.3 Motivating Example Revisited

Based on the motivating example of the introduction, we now express the queries mentioned there (see Figures 1, 2, 4, 5, 6) using the S-union, S-intersection and S-difference operators. For the presented examples, T_G is the tree of 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 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: $H_1 -^s (H_2 \cup^s H_3)$.
   According to Definition 3.1, $H_1 -^s (H_2 \cup^s H_3) = H_1 \cap^s (H_2 \cup^s H_3)'$. The nodes involved in $(H_2 \cup^s H_3)'$ are: caps, hoods, Close Up, UV, PL, film, B&W, slide. Figure 15 illustrates $H_2 \cup^s H_3$ as well as the final result.
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: $(H_1 -^s H_2) \cap^s (H_1 -^s H_3)$.
   Figure 16 shows the intermediate results $H_1 -^s H_2 = H_1 \cap^s H_2'$ and $H_1 -^s H_3 = H_1 \cap^s H_3'$, as well as the final result.

Notice that the result of the last two queries is the same, since the De Morgan's laws hold (see Table 1).

4 Lacking T_G

In the previous sections, we studied the algebraic properties of trees representing hierarchies of portal catalogs, based on the assumption that all of these trees are S-subsets of a tree available for the specific application domain or community, called global tree. Next, we present a framework to ensure that similar properties and laws hold in the absence of a global tree. Our target is to construct such a valid global tree T_G based on the available trees in such a way that all trees are S-subsets of that T_G.

The task resembles the S-union operation presented in Section 2.2 in the sense that the constructed T_G should provide integrated structural information from all available trees representing hierarchies of portal catalogs. Figure 17 shows trees T_1 and T_2 and the constructed

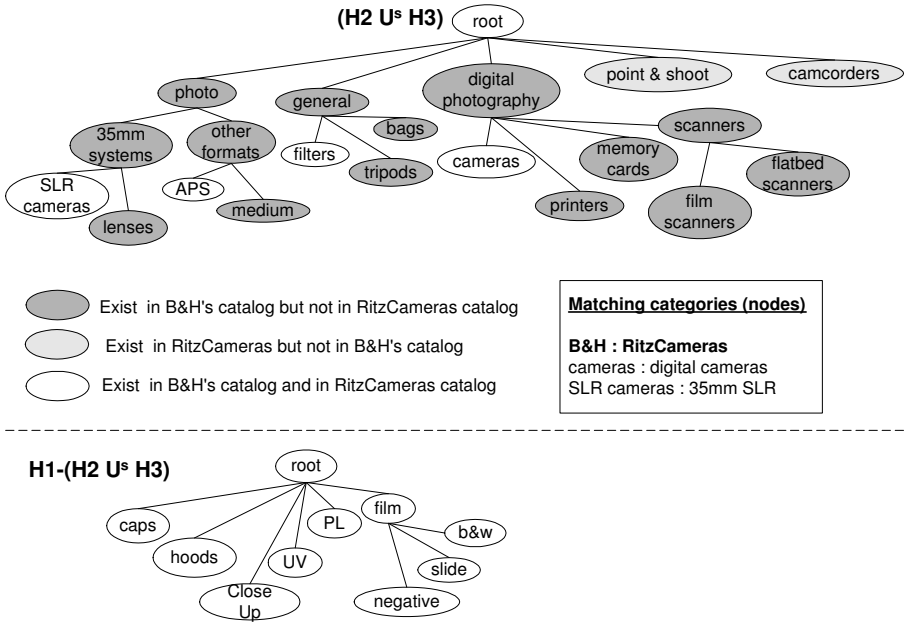

Figure 15: Example 1.

tree T_G. T_G contains all nodes from both T_1 and T_2. The structural relationships in T_3 are determined similarly to those determined in an S-union operation. However, contrary to the S-union operation, the construction of T_G is based on the structural relationships coming only from the available trees. The construction of T_G, given two trees T_i and T_j, is formalized as 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. The* global tree *of T_i and T_j is the tree $T_G = \langle \{r\} \cup N_G, \mathcal{P}_G \rangle$, constructed as follows:*

- $N_G = N_i \cup N_j$*:*
  T_G has all nodes from both T_i and T_j.
- $x\mathcal{P}_G y$ *if* $x\mathcal{P}_i y$ *and* $x\mathcal{P}_j y$*, with* $x, y \in \{r\} \cup N_G$*:*
  T_G keeps all parent relationships $p(x, y)$ common to both T_i and T_j.
- $x\mathcal{P}_G y$*, if* $x\mathcal{P}_i y$ *and* $\neg x\mathcal{P}_j^{tr} y$*, or if* $x\mathcal{P}_j y$ *and* $\neg x\mathcal{P}_i^{tr} y$ *($x, y \in \{r\} \cup N_G$):*
  T_G 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).

Consider now the trees T_2 and T_5 in Figure 18. During T_G construction, how one can handle $p(E, C)$ and $p(F, C)$ without any information about the structural relationship between E and F? See also the trees T_2 and T_3 in Figure 18, where $p(E, C)$ in T_2. Nodes E and C are children of the same node in T_2. Should we keep $p(E, C)$ or just consider E, C as children of the root of T_G? To deal with such issues, we introduce the notions of *consistency* and *semantic compatibility* for trees.

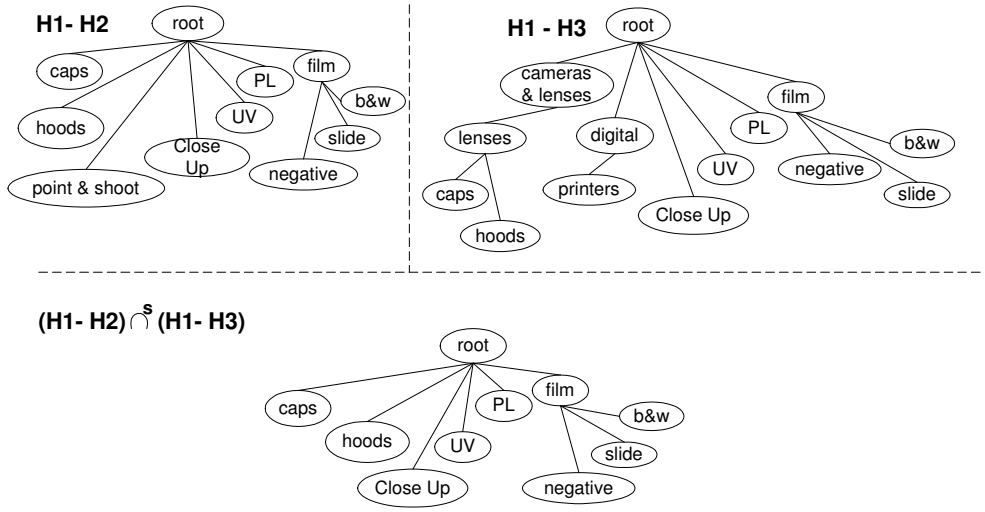

Figure 16: Example 2.

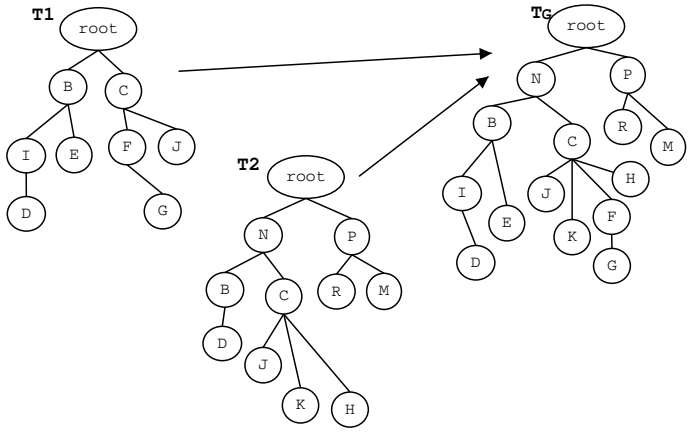

Figure 17: Trees T_1 and T_2 and the constructed tree T_G.

4.1 Consistency

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 18 are not consistent, since $p(E, C)$ in T_1, while E, C are children of the root of T_2. First we define consistency on a pair of trees, and then we expand the definition for a set of trees. The former is not enough for ensuring the construction of a T_G without structural conflicts. See the example of trees T_1, T_2 and T_3 in Figure 19. Constructing T_G^{23} from T_2 and T_3, and then T_G from T_1 and T_G^{23}, gives a T_G without structural conflicts. On the other hand, constructing T_G^{12} from T_1 and T_2, and then T_G from T_G^{12} and T_3, is vague due to nodes B and C. Intuitively, two trees are consistent if for every pair of their common nodes, the type of their relationship, regarding $\mathcal{P}^{tr}$ relation, is the same in both trees. That is if $\mathcal{P}^{tr}$ holds (does not hold) for such a pair in one tree, it should (not) hold in the other.

Definition 4.2. *Two trees* $T_i = \langle \{r\} \cup N_i, \mathcal{P}_i \rangle$ *and* $T_j = \langle \{r\} \cup N_j, \mathcal{P}_j \rangle$ *are* consistent *(or a pair*

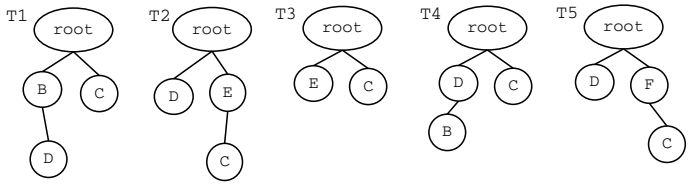

Figure 18: Trees to be examined for consistency and semantic compatibility.

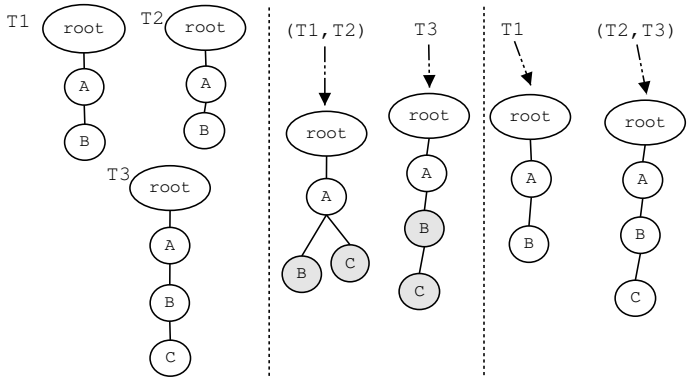

Figure 19: Constructing T_G given 3 trees: see the problem regarding shaded nodes B and C.

of trees is consistent*) if*

1. $\forall\, x, y \in \{r\} \cup (N_i \cap N_j)$ *with* $x\mathcal{P}_i^{tr} y$ *then* $x\mathcal{P}_j^{tr} y$*, and*
2. $\forall\, x, y \in \{r\} \cup (N_i \cap N_j)$ *with* $x\mathcal{P}_j^{tr} y$ *then* $x\mathcal{P}_i^{tr} y$

Trees T_1 and T_2 of Figure 18 have the pairs of nodes $(root, D)$, $(root, C)$ and (D, C) in common. Since $root\mathcal{P}_1^{tr} D$, $root\mathcal{P}_2^{tr} D$, $root\mathcal{P}_1^{tr} C$, $root\mathcal{P}_2^{tr} C$ hold, while $D\mathcal{P}_1^{tr} C$, $D\mathcal{P}_2^{tr} C$ not, T_1 and T_2 are consistent. Trees T_2 and T_3 of Figure 18 have the pairs $(root, E)$, $(root, C)$ and (E, C) in common. Since $E\mathcal{P}_2^{tr} C$ holds but $E\mathcal{P}_3^{tr} C$ not, T_2 and T_3 are inconsistent. According to the next proposition, two consistent trees are equal if they have the same nodes.

Proposition 4.1. *If two trees* $T_i = \langle \{r\} \cup N_i, \mathcal{P}_i \rangle$ *and* $T_j = \langle \{r\} \cup N_j, \mathcal{P}_j \rangle$ *are consistent and* $N_i = N_j$ *then* $T_i = T_j$*.*

Proof. Suppose that $T_i \neq T_j$. Then, since T_i, T_j are consistent, there exists a pair (x, y) for which $p(x, y)$ in T_i and $a(x, y)$ in T_j (results are the same for $p(x, y)$ in T_j and $a(x, y)$ in T_i). Therefore, there is a node z such that $x\mathcal{P}_j^{tr} z$ and $z\mathcal{P}_j^{tr} y$. Since both trees contain the same nodes, z should be in T_i, too. T_i and T_j are consistent, so both $x\mathcal{P}_i^{tr} z$ and $z\mathcal{P}_i^{tr} y$ should hold, but this cannot be the case because $p(x, y)$ in T_i. Thus, there is not a node z between x and y in T_j. So, for every pair (x, y) with $p(x, y)$ in T_i, $p(x, y)$ in T_j. Therefore, $T_i = T_j$. □

We next introduce the notion of *stability* for a pair of tree nodes, given a set of trees. Intuitively, stability ensures that new node relationships produced in the T_G will never be in conflict with already existing ones. Stability is required for the definition of consistency for a set of trees that will follow later on.

Definition 4.3. *Assume a set $\mathcal{S}$ of n trees, every pair of which is consistent, and the pair of nodes (x, y), with $x\mathcal{P}_i^{tr}y$ in one of the trees T_i in $\mathcal{S}$. We choose from $\mathcal{S}$ all the trees that contain exactly one of the nodes of the pair (either x or y but not both), and partition them in two sets: $\mathcal{G}_x$ is the set of trees that contain node x and $\mathcal{G}_y$ is the set of trees that contain node y. Iff for every pair of trees (T_x, T_y) with $T_x \in \mathcal{G}_x$ and $T_y \in \mathcal{G}_y$ there is a node z such that $x\mathcal{P}_x^{tr}z$ and $z\mathcal{P}_y^{tr}y$ then the pair of nodes (x, y), with $x\mathcal{P}_i^{tr}y$, is called* stable *in the set $\mathcal{S}$.*

We note that:

1. If $\mathcal{G}_x = \emptyset$ or $\mathcal{G}_y = \emptyset$ then x, y is stable in $\mathcal{S}$.
2. If a pair of nodes (x, y) is stable in $\mathcal{S}$, then $x\mathcal{P}_G^{tr}y$ holds in the T_G produced by any two trees of $\mathcal{S}$.
3. If a pair of nodes (x, y) is not stable in $\mathcal{S}$, then there exists at least one pair of trees in $\mathcal{S}$ for which the produced T_G has x, y without $x\mathcal{P}_G^{tr}y$.
4. If a pair of nodes (x, y) is stable in $\mathcal{S}$, then (x, y) is stable in any set $\mathcal{S}' \subseteq \mathcal{S}$.

Some examples follow:

1. Consider the set $\mathcal{S}'$ of trees T_1, T_2, T_3 and T_4 in Figure 20(a). Every pair of trees in $\mathcal{S}'$ is consistent and $B\mathcal{P}_1^{tr}D$. T_2 and T_3 both contain node D but not B, therefore $\mathcal{G}_D = \{T_2, T_3\}$. T_4 contains node B but not D, therefore $\mathcal{G}_B = \{T_4\}$. However, there is no node X such that $B\mathcal{P}_4^{tr}X$, $X\mathcal{P}_2^{tr}D$ and $x\mathcal{P}_3^{tr}D$, therefore (B, D) is not stable in $\mathcal{S}'$.
2. Consider the set $\mathcal{S}''$ of trees of trees T_1, T_2 and T_3 in Figure 20(b). Every pair of trees in $\mathcal{S}''$ is consistent and $A\mathcal{P}_1^{tr}B$. We put all trees that contain node A but not B in group $\mathcal{G}_A$, and all trees that contain node B but not A in $\mathcal{G}_B$. Therefore, $\mathcal{G}_A = \{T_2\}$ and $\mathcal{G}_B = \{T_3\}$. Tree T_2 in $\mathcal{G}_A$ has a node C for which $A\mathcal{P}_2^{tr}C$. Tree T_3 in $\mathcal{G}_B$ has the same node C for which $C\mathcal{P}_3^{tr}B$. Therefore (A, B) is stable in $\mathcal{S}''$.

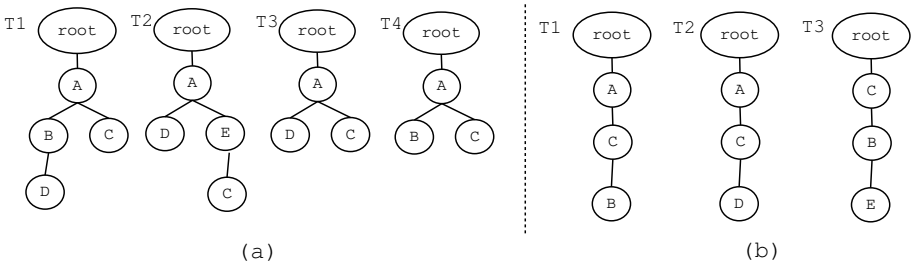

Figure 20: Example trees to check stability.

We next define the notion of *consistent* set of trees, that is pairwise consistent trees (see Definition 4.2) with only stable node pairs.

Definition 4.4. *Let $\mathcal{S}$ be a set of n trees and $\mathcal{N}$ the set of all nodes of all trees in $\mathcal{S}$. $\mathcal{S}$ is* consistent *if both the following conditions hold:*

1. *Every pair of trees in $\mathcal{S}$ is consistent.*

2. *Every pair of nodes (x, y), with $x\mathcal{P}^{tr}y$ in some tree in $\mathcal{S}$, is stable in $\mathcal{S}$ $(x, y \in \mathcal{N})$.*

We note that:

1. Since all trees under consideration have the same root, we omit all pairs containing the root to check the above condition, since these pairs are stable in any set of trees.
2. A set containing exactly 2 trees is always consistent if the two trees are consistent according to the definition 4.2.

Some examples follow:

1. Consider the set $\mathcal{S} = \{T_1, T_2, T_3\}$ in Figure 20(a). The set of all nodes is $\mathcal{N} = \{A, B, C, D, E\}$, and all the pairs of nodes, excluding those containing the root, are: (B, C), (B, D), (B, E), (C, D), (C, E), (D, E). From all these pairs, only $B\mathcal{P}^{tr}D$ and $C\mathcal{P}^{tr}E$ hold in some tree in $\mathcal{S}$. (B, D) and (C, E) are stable in $\mathcal{S}$, thus $\mathcal{S}$ is consistent.
2. On the other hand, $\mathcal{S}' = \{T_1, T_2, T_3, T_4\}$ (the full set of trees in Figure 20(a)) is not consistent, since (B, D) is not stable.

4.2 Semantic Compatibility

We next define semantic compatibility for a pair of trees. Semantic compatibility ensures that we will always be able to decide the structural relationship among the involved nodes in the new tree T_G. For example, trees T_2 and T_5 in Figure 18 are not semantically compatible, since it is not clear whether $p(E, C)$ or $p(F, C)$ in T_G.

Definition 4.5. *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 and $N = N_i \cap N_j$. T_i and T_j are* semantically compatible *if $\forall x \in N$, $\exists y$ with $p(y, x)$ in T_i or T_j with $y \in N$.*

In the following section we exploit the notions of tree consistency and tree semantic compatibility to study the validity of T_G construction.

4.3 Validity of Global Tree Construction

The construction of a valid T_G requires that the involved trees are consistent and every pair of the trees semantically compatible. We next present a set of theorems to support such an argument. As the following theorem states, the construction of T_G preserves the tree semantic compatibility in the tree set.

Theorem 4.1. *Let $\mathcal{S}$ be a set of trees, semantically compatible with each other. The tree $T_G = \langle\{r\} \cup N_G, \mathcal{P}_G\rangle$ constructed from any two trees $T_i = \langle\{r\} \cup N_i, \mathcal{P}_i\rangle$ and $T_j = \langle\{r\} \cup N_j, \mathcal{P}_j\rangle$ in $\mathcal{S}$ is semantically compatible with all other trees of $\mathcal{S}$.*

Proof. Let $T_k = \langle\{r\} \cup N_k, \mathcal{P}_k\rangle$ a tree in $\mathcal{S}$. Since T_i and T_k are semantically compatible, $\forall x \in N_i \cap N_k$, $\exists y$ with $p(y, x)$ in T_i or T_k, $y \in N_i \cap N_k$. Similarly, since T_j and T_k are semantically compatible, $\forall x \in N_j \cap N_k$, $\exists y$ with $p(y, x)$ in T_j or T_k, $y \in N_j \cap N_k$. The parent node of a node x in T_G will be the same with the parent node of x in T_i or in T_j. Thus, $\forall x \in (N_i \cap N_k) \cup (N_j \cap N_k) = (N_i \cup N_j) \cap N_k = N_G \cap N_k$, $\exists y$ with $p(y, x)$ in T_G or T_k, $y \in (N_i \cap N_k) \cup (N_j \cap N_k) = N_G \cap N_k$, that is T_G is semantically compatible with T_K. □

The construction of T_G should preserve tree consistency in the tree set. To simplify the proof of the related theorem, we first present 3 relevant lemmas.

Lemma 4.1. *Let $\mathcal{S}$ be a consistent set of n trees. The tree T_G constructed from any two trees in $\mathcal{S}$ will produce a set $\mathcal{S}'$ of $(n-1)$ trees, in which every two trees are consistent.*

Proof. Consider the sets $\mathcal{S} = \{T_1, T_2, T_3, \ldots, T_n\}$ and $\mathcal{S}' = \{T_G, T_3, \ldots, T_n\}$, where T_G is constructed from T_1 and T_2 (results are similar for any other pair of trees). Since $\mathcal{S}$ is consistent, every pair of trees in $\mathcal{S}$ is consistent, and every pair of nodes satisfying the $\mathcal{P}^{tr}$ relation in some tree in $\mathcal{S}$ is stable in $\mathcal{S}$. We will show that T_G is consistent with every tree in set $\mathcal{R} = \{T_3, \ldots, T_n\}$. Consider a pair of nodes (x, y) in tree T_G. We assume that there is at least one tree T_i in $\mathcal{R}$ that contains both x and y. Otherwise, there would be no inconsistency caused by this pair. There are two possibilities:

1. $x, y \in T_1$ (results are similar for $x, y \in T_2$): Since T_1 is consistent with every tree T_i in $\mathcal{R}$, the kind of the relationship between x and y (regarding $\mathcal{P}^{tr}$) in T_i is the same with that in T_1, and, thus, the same with that in T_G, since T_G is constructed from T_1 and T_2.

2. $x \in T_1$ and $y \in T_2$ (results are similar for $x \in T_2$ and $y \in T_1$):

   (a) $x\mathcal{P}^{tr}_G y$ holds:
   There exists a node z, common in T_1 and T_2, such that $x\mathcal{P}^{tr}_1 z$ and $z\mathcal{P}^{tr}_2 y$.
   i. If $z \in T_i \Longrightarrow x\mathcal{P}^{tr}_i z$ and $z\mathcal{P}^{tr}_i y$ (since T_i is consistent with T_1 and T_2) $\Longrightarrow x\mathcal{P}^{tr}_i y$
   ii. If $z \notin T_i$: $\mathcal{G}_x = \{T_i, \ldots\}$ (since T_i contains x but not z) and $\mathcal{G}_z = \{T_2, \ldots\}$ (since T_2 contains z but not x). Since $\mathcal{S}$ is consistent, (x, z) is stable in $\mathcal{S} \Longrightarrow$
   $\exists p : x\mathcal{P}^{tr}_i p$ and $p\mathcal{P}^{tr}_2 z \Longrightarrow \exists p : x\mathcal{P}^{tr}_i p$ and $p\mathcal{P}^{tr}_2 y$ (since $z\mathcal{P}^{tr}_2 y$ holds) $\Longrightarrow$
   $\exists p : x\mathcal{P}^{tr}_i p$ and $p\mathcal{P}^{tr}_i y$ (since T_i is consistent with T_2) $\Longrightarrow x\mathcal{P}^{tr}_i y$.

   (b) $x\mathcal{P}^{tr}_G y$ does not hold:
   Suppose that $x\mathcal{P}^{tr}_i y$. $\mathcal{G}_x = \{T_1, \ldots\}$ (since T_1 contains x but not y) and $\mathcal{G}_y = \{T_2, \ldots\}$ (since T_2 contains y but not x). (x, y) is stable in $\mathcal{S} \Longrightarrow$
   $\exists z : x\mathcal{P}^{tr}_1 z$ and $z\mathcal{P}^{tr}_2 y \Longrightarrow x\mathcal{P}^{tr}_G y$ holds (since T_G is constructed from T_1 and T_2), but we assumed otherwise. Therefore $x\mathcal{P}^{tr}_i y$ does not hold.

□

Lemma 4.2. *Let $\mathcal{S}$ be a set of trees and $\mathcal{N}$ the set of all nodes of all trees in $\mathcal{S}$. If $\mathcal{S}$ is consistent then every pair of nodes $x, y \in \mathcal{N}$, satisfying the $\mathcal{P}^{tr}$ relation in some tree in $\mathcal{S}$, is stable in the set $\mathcal{S}' = \mathcal{S} \cup \{T_G\}$, where T_G is constructed from any two trees of $\mathcal{S}$.*

Proof. We consider that T_G is constructed from two trees $T_1, T_2 \in \mathcal{S}$ (results are similar for any other pair of trees). Every pair of nodes x, y, satisfying the $\mathcal{P}^{tr}$ relation in some tree in $\mathcal{S}$, is stable in $\mathcal{S}$, so there exist $\mathcal{G}_x$ (the set of trees in $\mathcal{S}$ with x but not y) and $\mathcal{G}_y$ (the set of trees $\mathcal{S}$ with y but not x). We will check the stability of (x, y) in $\mathcal{S}'$.

1. If $x\mathcal{P}^{tr}_G y$ or $x, y \notin T_G$, then there is no change in $\mathcal{G}_x$ and $\mathcal{G}_y$, thus (x, y) is stable in $\mathcal{S}'$.

2. If $x \in T_G$ and $y \notin T_G$ (results are similar for $y \in T_G$ and $x \notin T_G$), then $\mathcal{G}'_x = \mathcal{G}_x \cup \{T_G\}$ and $\mathcal{G}'_y = \mathcal{G}_y$. Since $x \in T_G$ and $y \notin T_G$, it must be $x \in T_1$ and $y \notin T_1$ (results are similar for $x \in T_2$ and $y \notin T_2$), thus $T_1 \in \mathcal{G}'_x$. (x, y) is stable in $\mathcal{S}$, so for every pair (T_1, T_y),

$T_y \in \mathcal{G}'_y$, there exists a node z such that $x\mathcal{P}_1^{tr}z$ and $z\mathcal{P}_y^{tr}y$. Consequently, for every pair (T_G, T_y), $T_y \in \mathcal{G}'_y$, there exists a node z such that $x\mathcal{P}_G^{tr}z$ and $z\mathcal{P}_y^{tr}y$, because T_G contains all structural information of tree T_1. Therefore, (x, y) is stable in $\mathcal{S}'$.

Note also that (x, y) is stable in set $\mathcal{S}'' = \mathcal{S} \cup \{T_G\} - \{T_1, T_2\}$, because $\mathcal{S}'' \subset \mathcal{S}'$. □

Lemma 4.3. *Let $\mathcal{S}$ be a consistent set of 4 trees. The tree T_G constructed from any two trees in $\mathcal{S}$ results in a set $\mathcal{S}'$ with 3 trees which is also consistent.*

Proof. Consider $\mathcal{S} = \{T_1, T_2, T_3, T_4\}$ and $\mathcal{S}' = \{T_G, T_3, T_4\}$, where T_G is constructed from T_1 and T_2 (results are similar for any other pair of trees). Every pair of trees in $\mathcal{S}$ is consistent (Lemma 4.1). Every pair of nodes x, y, satisfying the $\mathcal{P}^{tr}$ relation in some tree in $\mathcal{S}$, is stable in set $\mathcal{S}'$ (Lemma 4.2). Therefore, we should check the stability of a pair of nodes x, y, $x\mathcal{P}_G^{tr}y$, such that $x\mathcal{P}^{tr}y$ does not hold in some tree in $\mathcal{S}$. $x\mathcal{P}_1^{tr}y$ does not hold, nor $x\mathcal{P}_2^{tr}y$, so $x \in T_1$ and $y \in T_2$ (results are similar for $y \in T_1$ and $x \in T_2$).

For $x\mathcal{P}_G^{tr}y$ to hold, $\exists z : x\mathcal{P}_1^{tr}z$ and $z\mathcal{P}_2^{tr}y$. If $x \notin T_3, T_4$ or $y \notin T_3, T_4$, than (x, y) is stable in $\mathcal{S}'$. We assume that $x \in T_3$ and $y \in T_4$ (since $x\mathcal{P}^{tr}y$ does not hold in some tree in $\mathcal{S}$). Checking the stability of (x, y), with $x\mathcal{P}^{tr}y$ in some tree of $\mathcal{S}$, in $\mathcal{S}'$ results[2] in: $\mathcal{G}_x = \{T_3\}$ and $\mathcal{G}_y = \{T_4\}$. Only if there is a node p such that $x\mathcal{P}_3^{tr}p$ and $p\mathcal{P}_4^{tr}y$, will (x, y) will be stable in $\mathcal{S}'$. We will look for such a node p.

1. $z \in T_3$ and $z \in T_4$: $p = z$
2. $z \notin T_3$ and $z \notin T_4$: since (x, z) is stable in $\mathcal{S}$, $\mathcal{G}_x = \{T_3\}$ and $\mathcal{G}_z = \{T_2\}$. Thus, $\exists w : x\mathcal{P}_3^{tr}w$ and $w\mathcal{P}_2^{tr}z \implies w\mathcal{P}_2^{tr}y$ (since $z\mathcal{P}_2^{tr}y$ holds).
   (a) If $w \in T_4$ then $p = w$.
   (b) If $w \notin T_4$, then, since (w, y) is stable in $\mathcal{S}$, $\mathcal{G}_w = \{T_3\}$ and $\mathcal{G}_y = \{T_4\}$ (results are similar if $T_1 \in \mathcal{G}_w$ or $T_1 \in \mathcal{G}_y$). Thus, $\exists f : w\mathcal{P}_3^{tr}f$ and $f\mathcal{P}_4^{tr}y \implies \exists f : x\mathcal{P}_3^{tr}f$ and $f\mathcal{P}_4^{tr}y$ (since $x\mathcal{P}_3^{tr}w$ holds) $\implies p = f$.
3. $z \in T_3$ and $z \notin T_4$ (results are similar for $z \in T_4$ and $z \notin T_3$): since (z, y) is stable in $\mathcal{S}$, $\mathcal{G}_z = \{T_1, T_3\}$ and $\mathcal{G}_y = \{T_4\}$. Thus $\exists w : z\mathcal{P}_3^{tr}w$ and $w\mathcal{P}_4^{tr}y \implies \exists w : x\mathcal{P}_3^{tr}w$ and $w\mathcal{P}_4^{tr}y$ $\implies p = w$.

Therefore, (x, y) is stable in $\mathcal{S}'$, hence $\mathcal{S}'$ is consistent. □

We next prove that the construction of T_G preserves tree consistency in the tree set.

Theorem 4.2. *Let $\mathcal{S}$ be a consistent set of n trees. The tree T_G constructed from any two trees in $\mathcal{S}$ results in a set $\mathcal{S}'$ with $n-1$ trees which is also consistent.*

Proof. Theorem holds for $n = 4$ (Lemma 4.3). Given that it holds for $n = k$, too, we will prove that it holds for $n = k + 1$. Assume the consistent sets $\mathcal{S}_k = \{T_1, T_2, T_3, \ldots, T_k\}$ and $\mathcal{S}'_k = \{T_G, T_3, \ldots, T_k\}$, where T_G is constructed from T_1 and T_2 (results are similar for any other pair of trees). Let $\mathcal{R} = \{T_3, \ldots, T_k\}$, so $\mathcal{S}_k = \{T_1, T_2\} \cup \mathcal{R}$ and $\mathcal{S}'_k = \{T_G\} \cup \mathcal{R}$.

Consider now the set $\mathcal{S}_{k+1} = \{T_1, T_2, \ldots, T_k, T_{k+1}\} = \{T_1, T_2\} \cup \mathcal{R} \cup \{T_{k+1}\}$, which is consistent. We construct a global tree using two trees from $\mathcal{S}_{k+1}$, getting the set $\mathcal{S}'_{k+1}$. There are 2 options for such a construction:

[2]According to Definition 4.3, if a pair of nodes x, y is stable in $\mathcal{S}$, then we are able to identify the two sets $\mathcal{G}_x, \mathcal{G}_y$.

1. $\mathcal{S}'_{k+1} = \{T_G, T_{k+1}\} \cup \mathcal{R}$, where T_G is constructed from T_1 and T_2 (Case 1).
2. $\mathcal{S}'_{k+1} = \{T'_G, T_2\} \cup \mathcal{R}$, where T'_G is constructed from T_1 and T_{k+1} (Case 2).

We will show that $\mathcal{S}'_{k+1}$ is consistent.

The first condition of definition 4.4 is satisfied for both 1. and 2. (Lemma 4.1). Moreover, every pair of nodes x, y, with $x\mathcal{P}^{tr}_y$ in some tree of $\mathcal{S}_{k+1}$, is stable in $\mathcal{S}'_{k+1}$ (Lemma 4.2). Therefore, we only have to check the stability of (x, y) in $\mathcal{S}'_{k+1}$, with $x\mathcal{P}^{tr}_G y$ (if in Case 1) or $x\mathcal{P}^{tr}_{G'} y$ (if in Case 2), which does not exist in set $\mathcal{S}_{k+1}$ but exists in $\mathcal{S}'_{k+1}$. Let $\mathcal{R}_x = \{T \in \mathcal{R}, x \in T, y \notin T\}$, that is the subset of $\mathcal{R}$ with trees containing only node x and not y, and $\mathcal{R}_y = \{T \in \mathcal{R}, y \in T, x \notin T\}$, that is the subset of $\mathcal{R}$ with trees containing only node y and not x.

Case 1 ($\mathcal{S}'_{k+1} = \{T_G, T_{k+1}\} \cup \mathcal{R}$, T_G is constructed from T_1 and T_2)

$x, y \in T_G$. Since (x, y) is stable in $\mathcal{S}'_k$, we can identify the sets $\mathcal{G}_x, \mathcal{G}_y$ (see Definition 4.3): $\mathcal{G}_x = \mathcal{R}_x$ and $\mathcal{G}_y = \mathcal{R}_y$.

1. If $x, y \in T_{k+1}$ or ($x \notin T_{k+1}$ and $y \notin T_{k+1}$) then checking the stability of (x, y) in $\mathcal{S}'_{k+1}$ gives $\mathcal{G}_x = \mathcal{R}_x$ and $\mathcal{G}_y = \mathcal{R}_y$ (that is $\mathcal{G}_x, \mathcal{G}_y$ do not change), thus (x, y) stable in $\mathcal{S}'_{k+1}$.
2. If $x \in T_{k+1}$ and $y \notin T_{k+1}$ (results are similar for $y \in T_{k+1}$ and $x \notin T_{k+1}$), then checking the stability of (x, y) in $\mathcal{S}'_{k+1}$ gives $\mathcal{G}_x = \mathcal{R}_x \cup \{T_{k+1}\}$ and $\mathcal{G}_y = \mathcal{R}_y$.
   (a) $\mathcal{R}_x \neq \emptyset$ and $\mathcal{R}_y \neq \emptyset$
   For every pair (T_x, T_y) , $T_x \in \mathcal{R}_x$ and $T_y \in \mathcal{R}_y$, $\exists c : x\mathcal{P}^{tr}_x c$ and $c\mathcal{P}^{tr}_y y$, because (x, y) is stable in $\mathcal{S}'_k$. If T_{k+1} contains all these nodes c, then (x, y) is stable in $\mathcal{S}'_{k+1}$, too. If there is a node $c \notin T_{k+1}$, then $T_{k+1} \in \mathcal{G}_x$ and $\mathcal{R}'_y \subseteq \mathcal{G}_c$, where $\mathcal{R}'_y = \{T \in \mathcal{R}_y, c \in T\}$ (that is the subset of $\mathcal{R}_y$ that contains c), since (x, c) is stable in $\mathcal{S}_{k+1}$. Therefore, for every pair (T_{k+1}, T_y), $T_y \in \mathcal{R}'_y$, $\exists d : x\mathcal{P}^{tr}_{k+1} d$ and $d\mathcal{P}^{tr}_y c \implies \exists d : x\mathcal{P}^{tr}_{k+1} d$ and $d\mathcal{P}^{tr}_y y$ (since $c\mathcal{P}^{tr}_y y$ holds). So, (x, y) is stable in $\mathcal{S}'_{k+1}$.
   (b) $\mathcal{R}_y = \emptyset$ then (x, y) is stable in $\mathcal{S}'_{k+1}$ (see the notes after Definition 4.3).
   (c) $\mathcal{R}_x = \emptyset$
   Since $x\mathcal{P}^{tr}_G y$ holds, $\exists c : x\mathcal{P}^{tr}_1 c$ and $c\mathcal{P}^{tr}_2 y$. Let $\mathcal{R}_1 = \{T : T \in \mathcal{R}_y \text{ and } \in T\}$ and $\mathcal{R}_2 = \{T : T \in \mathcal{R}_y \; c \notin T\}$, with $\mathcal{R}_y = \mathcal{R}_1 \cup \mathcal{R}_2$.
      i. $c \in T_{k+1}$
      Since (c, y) is stable in $\mathcal{S}_{k+1}$, $\mathcal{G}_c = \{T_1, T_{k+1}\}$ and $\mathcal{G}_y = R_2$, thus for every pair (T_{k+1}, T), $T \in R_2$, $\exists d : c\mathcal{P}^{tr}_{k+1} d$ and $d\mathcal{P}^{tr} y \implies \exists d : x\mathcal{P}^{tr}_{k+1} d$ and $d\mathcal{P}^{tr} y$ (since $x\mathcal{P}^{tr}_{k+1} c$ holds). We also have that $x\mathcal{P}^{tr}_{k+1} c$ and $c\mathcal{P}^{tr} y$ in every $T \in R_1$, therefore (x, y) is stable in $\mathcal{S}'_{k+1}$.
      ii. $c \notin T_{k+1}$
      Since (x, c) is stable in $\mathcal{S}_{k+1}$, $\mathcal{G}_x = \{T_{k+1}\}$ and $\mathcal{G}_c = R_1 \cup \{T_2\} \implies$ for every pair (T_{k+1}, T), $T \in R_1$, $\exists d : x\mathcal{P}^{tr}_{k+1} d$ and $d\mathcal{P}^{tr} c \implies \exists d : x\mathcal{P}^{tr}_{k+1} d$ and $d\mathcal{P}^{tr} y$ (since $c\mathcal{P}^{tr} y$ holds). Given that (c, y) is stable in $\mathcal{S}_{k+1}$, we have $\mathcal{G}_c = \{T_1\}$ and $\mathcal{G}_y = R_2 \implies$ for every pair (T_1, T), $T \in R_2$, $\exists e : c\mathcal{P}^{tr}_1 e$ and $e\mathcal{P}^{tr} y \implies \exists e : x\mathcal{P}^{tr}_1 e$ and $e\mathcal{P}^{tr} y$ (since $x\mathcal{P}^{tr}_1 c$ holds).
         A. If $e \in T_{k+1}$ then (x, y) is stable in $\mathcal{S}'_{k+1}$.

B. If $e \notin T_{k+1}$, then, since (x, e) is stable in $\mathcal{S}_{k+1}$, we have $\mathcal{G}_x = \{T_{k+1}\}$ and $\mathcal{G}_e = \mathcal{R}_2$, so for every pair (T_{k+1}, T), $T \in \mathcal{R}_1$, $\exists f : x\mathcal{P}^{tr}_{k+1} f$ and $f\mathcal{P}^{tr} e \Longrightarrow$ $\exists f : x\mathcal{P}^{tr}_{k+1} f$ and $f\mathcal{P}^{tr} y$ (since $e\mathcal{P}^{tr} y$ holds).

Therefore, for every pair (T_{k+1}, T), with $T \in \mathcal{R}_y$, there is a node o ($o = d$,e or f) such that $x\mathcal{P}^{tr}_{k+1} o$ and $o\mathcal{P}^{tr} y$, so (x, y) is stable in $\mathcal{S}'_{k+1}$.

3. If $y \in T_{k+1}$ and $x \notin T_{k+1}$ then the case is symmetrical to the previous one and it is handled the same way.

Case 2 ($\mathcal{S}'_{k+1} = \{T'_G, T_2\} \cup \mathcal{R}$, T'_G is constructed from T_1 and T_{k+1})

$\mathcal{S}'_{k+1} = \{T'_G\} \cup \mathcal{V}$, where $\mathcal{V} = \mathcal{R} \cup \{T_2\}$, and T'_G is constructed from T_1 and T_{k+1}. Checking the stability of (x, y) in $\mathcal{S}'_{k+1}$ gives $\mathcal{G}_x = \mathcal{V}_x$ and $\mathcal{G}_y = \mathcal{V}_y$, where $\mathcal{V}_x$ is the subset of $\mathcal{V}$ containing node x and not y, and $\mathcal{V}_y$ is the subset of $\mathcal{V}$ containing node y and not x. If $\mathcal{V}_x = \emptyset$ or $\mathcal{V}_y = \emptyset$, then (x, y) is stable in $\mathcal{S}'_{k+1}$ (see notes after Definition 4.3). If $\mathcal{V}_x \neq \emptyset$ and $\mathcal{V}_y \neq \emptyset$, we can assume that $x \in T_1$ and $y \in T_{k+1}$. Since $x\mathcal{P}^{tr}_{G'} y$ holds, $\exists c : x\mathcal{P}^{tr}_1 c$ and $c\mathcal{P}^{tr}_{k+1} y$. Given that (c, y) is stable in $\mathcal{S}_{k+1}$, we have:

1. $T_1 \in \mathcal{G}_c$ and $\mathcal{V}'_y \in \mathcal{G}_y$, where $\mathcal{V}'_y$ is the subset of $\mathcal{V}_y$ that does not contain node c, and
2. for every pair (T_1, T), $T \in \mathcal{V}'_y$, $\exists d : c\mathcal{P}^{tr}_1 d$ and $d\mathcal{P}^{tr} y$.

For any T in $\mathcal{V}_y$, $x\mathcal{P}^{tr}_G y$ holds in T_G constructed from T_1 and T.

We have assumed that this theorem holds for $\mathcal{S}_k$, that is the set $\mathcal{S}'_k$ produced after the construction of T_G using T_1 and T_2 of $\mathcal{S}_k$ should be consistent. Thus, (x, y) is stable in the set $\mathcal{S}'_k = \{T_G\} \cup R$, so we can identify the sets $\mathcal{G}_x, \mathcal{G}_y$: $\mathcal{G}_x = \mathcal{R}_x$ and $\mathcal{G}_y = \mathcal{R}_y$. For every pair (T_x, T_y), $T_x \in \mathcal{R}_x$ and $T_y \in \mathcal{R}_y$, $\exists d : x\mathcal{P}^{tr}_x d$ and $d\mathcal{P}^{tr}_y y$. It is $\mathcal{V}_x = \mathcal{R}_x$ and $\mathcal{V}_y = \mathcal{R}_y \cup \{T_2\}$. Therefore, if $d \in T_2$, then (x, y) is stable in $\mathcal{S}'_{k+1}$. If $d \notin T_2$, then, since (d, y) is stable in $\mathcal{S}_{k+1}$, we have $\mathcal{R}'_x \subseteq \mathcal{G}_d$ and $T_2 \in \mathcal{G}_y$, where $\mathcal{R}'_x$ is the subset of $\mathcal{R}_x$ that contains d. Thus, for every pair (T, T_2) where $T \in \mathcal{R}'_x$, $\exists f : d\mathcal{P}^{tr} f$ and $f\mathcal{P}^{tr}_2 y \Longrightarrow \exists f : x\mathcal{P}^{tr} f$ and $f\mathcal{P}^{tr}_2 y$ (since $x\mathcal{P}^{tr} d$ holds).

So, for every pair (T, T_2), where $T \in \mathcal{R}_x$, there exists a node p ($p = d$ or f) such that $x\mathcal{P}^{tr} p$ and $p\mathcal{P}^{tr}_2 y$. Thus, (x, y) is stable in $\mathcal{S}'_{k+1}$.

In every case (x, y) is stable in $\mathcal{S}'_{k+1}$, so $\mathcal{S}'_{k+1}$ is consistent. □

There is still the question whether there are non-consistent sets of trees for which the construction of T_G makes the set consistent. The next theorem shows that Definition 4.4 for consistent set of trees covers all tree sets for which T_G construction produces consistent tree sets.

Theorem 4.3. *Let $\mathcal{S}$ be a set of n trees, every pair of which is consistent. The tree T_G constructed from any two trees in $\mathcal{S}$ results in a set $\mathcal{S}'$ with $n - 1$ trees. If every pair of trees in $\mathcal{S}'$ is consistent, then both $\mathcal{S}$ and $\mathcal{S}'$ are consistent.*

Proof. Let $\mathcal{N}$ be the set of all nodes of all trees in $\mathcal{S}$. Suppose that $\mathcal{S}$ is not consistent. Lack of stability for (x, y) means that there is no node c such that $x\mathcal{P}^{tr}_1 c$ and $c\mathcal{P}^{tr}_2 y$, given two trees T_1, T_2 in $\mathcal{S}$ with $x \in T_1$, $y \notin T_1$, $x \notin T_2$ and $y \in T_2$. Since the set of x's descendants in T_1 and the set of y's ancestors in T_2 have no nodes in common, then $x\mathcal{P}^{tr}_G y$ does not hold in T_G constructed from T_1 and T_2. Let T_3 be a tree in $\mathcal{S}$ with $x\mathcal{P}^{tr}_3 y$. T_3 also belongs to set $\mathcal{S}'$ and T_3 is not consistent with T_G. So, if $\mathcal{S}$ is not consistent, then the pairs of trees in $\mathcal{S}'$ are not

consistent. Consequently, if the pairs of trees in $\mathcal{S}'$ are consistent, then $\mathcal{S}$ should be consistent, and, thus, $\mathcal{S}'$ should be consistent, too (Theorem 4.2). □

The construction of T_G according to Definition 4.1 is a binary operation in the sense that 2 trees are used to build T_G. Having k trees, $k > 2$, say $\{T_1, T_2, T_3, \ldots, T_k\}$, one can build a T_G^{12} from T_1 and T_2, then a T_G^{123} from T_G^{12} and T_3, ..., and finally the T_G from $T_G^{12\ldots(k-1)}$ and T_k. The T_G constructed from many trees should be the same regardless the sequence of the operation. The next theorem ensures that requirement.

Theorem 4.4. *Let $\mathcal{S}$ be a consistent set of n trees. The tree T_G constructed from all trees is of $\mathcal{S}$ is the same, regardless the sequence in which the construction is performed.*

Proof. Let $\mathcal{N}$ be the set of all nodes of all trees in $\mathcal{S}$. Consider that constructing T_G in $\mathcal{S}$ in a certain order produces tree T_G^1, while in a different order produces tree T_G^2. We will show that $T_G^1 = T_G^2 = T_G$. $\mathcal{S}' = \mathcal{S} \cup \{T_G^1\}$ is consistent because T_G^1 is consistent with every tree in $\mathcal{S}$, and every pair of nodes (x, y), $x, y \in \mathcal{N}$, is stable in $\mathcal{S}'$ (T_G^1 contains all nodes of $\mathcal{S}$, so it does not affect the sets $\mathcal{G}_x, \mathcal{G}_y$ - see Definition 4.3). Since $\mathcal{S}'$ is consistent, any T_G construction will result in a consistent set, as shown in Theorem 4.2. Therefore, we can perform repeated T_G constructions in $\mathcal{S}$ so that tree T_G^2 is produced. After these constructions, $\mathcal{S}' = \{T_G^2\} \cup \{T_G^1\}$ (still consistent). Since T_G^1 and T_G^2 are consistent and contain the same set of nodes (which is the set of all nodes in $\mathcal{S}$), $T_G^1 = T_G^2$ (Proposition 4.1). □

5 Manipulating Hierarchical Schemas for the Web: a Demonstration in the Case of RDF/S Tree-like Hierarchies

We have applied our framework for manipulating hierarchical schemas on tree-like hierarchies encoded as RDF/s files. We use RDFSculpt [14], a prototype system that support integration of hierarchies based on union, intersection and difference semantics. The system implements the S-union, S-intersection and S-difference operators, and provides tools to explore and visualize hierarchies.

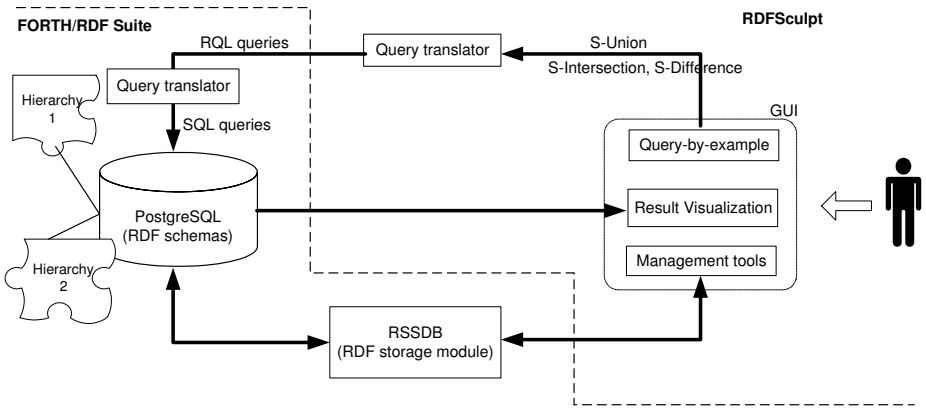

Figure 21: The architecture of RDFSculpt

As shown in Figure 21, the system is built on top of the ICS-FORTH RDFSuite[3]. To this extend, it exploits all RDF management and query APIs offered by RDFSuite. Results are visualized using RDFSViz[4]. Users can issue queries on hierarchies and produce new, integrated ones, using S-union, S-intersection and S-difference operators. The RDFSculpt assists the user in queries formulation, offering her query-by-example capabilities.

Assume that the user needs to find the part of Adorama's (H_1) catalog which is not present in the merged catalog produced from B&H's (H_2) and RitzCameras (H_3) catalogs (see Figure 1). The query is expressed as follows:

$$H_1 -^s (H_2 \cup^s H_3)$$

Figure 22 shows how the user can apply the S-union operator for the B&H's (H_2) and RitzCameras (H_3) catalogs, using the querying interface of our system.

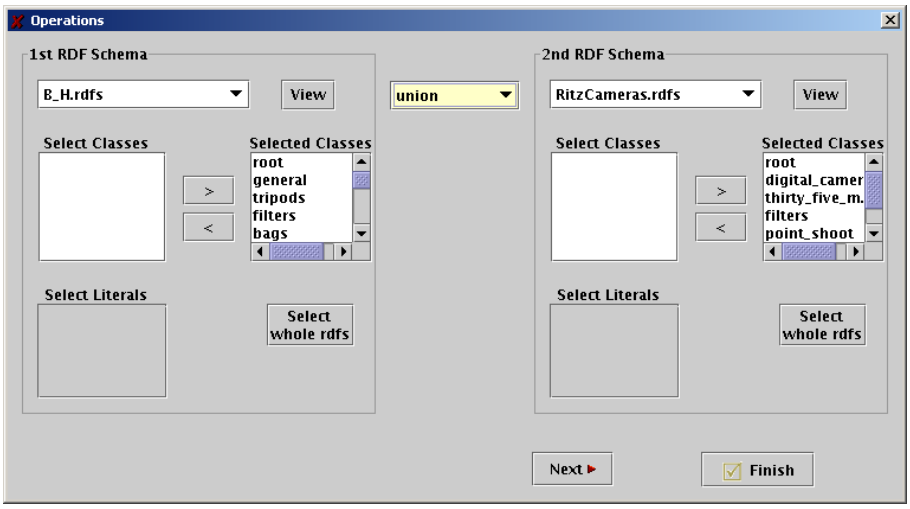

Figure 22: Merging B&H's and RitzCameras catalogs using the $S-$union operator.

Merging B&H's and RitzCameras catalogs using the $S-$union operator results to a catalog named $R1$. The user can now apply the S-difference operator for Adorama's and $R1$'s catalogs, as Figure 23 shows. At any time, the user can visualize any intermediate results. The final result of the query is shown in Figure 24.

6 Conclusions and Further Work

This work presented a framework for the structural manipulation of hierarchies in portal catalogs. It supports the integration of hierarchies based on union, intersection and difference semantics provided by a set of operators applied on hierarchies as full-fledged objects and not only as sets of nodes.

Specifically, we defined three set-like query operators applied on trees representing hierarchies: S-union, S-intersection and S-difference. The S-union operator provides an integrated

[3]http://www.ics.forth.gr/isl/RDF/index.html
[4]http://www.dfki.uni-kl.de/frodo/RDFSViz/

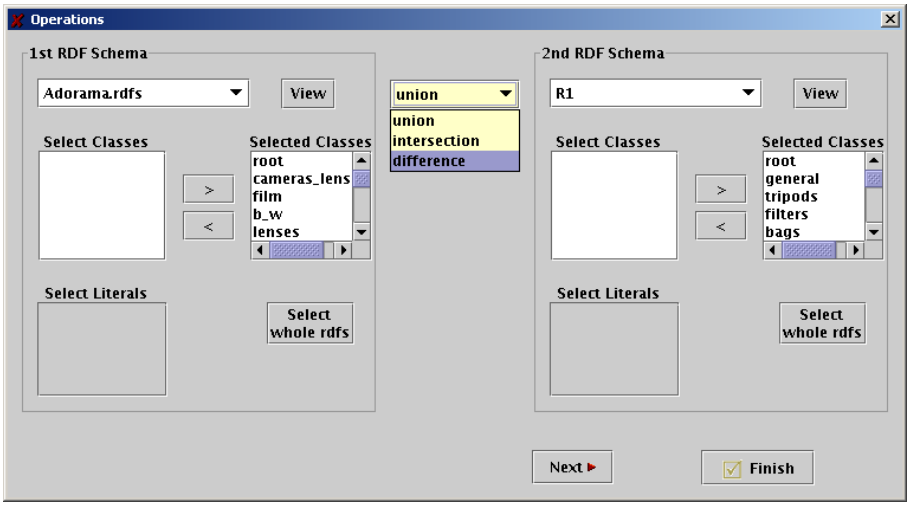

Figure 23: Finding the part of Adorama's catalog not present in R1.

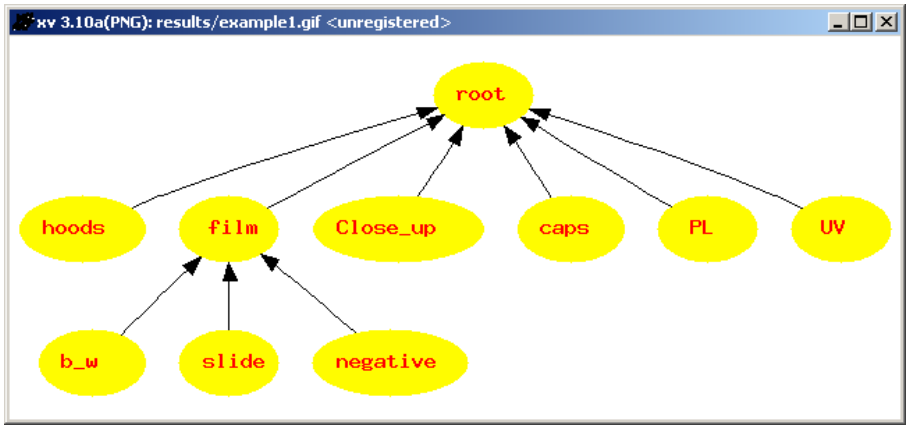

Figure 24: Final result of the first example

form of all structural information present in the involved hierarchies. The S-intersection operator provides the common structural information present in the involved hierarchies. The S-difference operator provides structural information present in one hierarchy but not in the other. The operators are based on the upper bound, lower bound and complement, respectively, of a boolean lattice algebraic structure defined on trees. That structure ensures that the operators have certain algebraic properties to provide clear semantics and assist the transformation and optimization of sequences of operations, using laws similar to those in set theory. Our framework was initially developed with the assumption that the involved trees have certain properties with respect to a given tree called global tree. To ensure that similar algebraic properties and laws hold in the absence of a global tree, we studied the conditions under which such a tree can be constructed using the available trees. Finally, we showed examples of using such operators to manipulate hierarchies of portal catalogs encoded as RDF/s hierarchies.

Our future work is based on three directions. First, we will further explore the algebraic properties of trees representing hierarchies in the absence of a global tree. Even if there are

structural inconsistencies in the involved trees during the construction of a global tree, one can decide to resolve these inconsistencies in such a way that all involved trees are S-subsets of some global tree. We will study how such an extension affects our framework. Furthermore, we will study how a similar framework can be established, having graphs instead of trees as the underline model to capture hierarchies with complex structure. Finally, we plan to employ schema matching techniques to deal with hierarchies with naming disimilarities (different naming for semantically similar categories).

References

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

[2] S. Abiteboul, P. Buneman, and D. Suciu, *Data on the web. from relations to semistructured data and xml*, Morgan Kaufmann Publishers, San Francisco, California, 2000.

[3] F. Bancilhon and S. Khoshafian, *A calculus for complex objects*, Proceedings of ACM Symposium on Principles of Database Systems (PODS'86), Cambridge, Massachusetts, Mar 1986, pp. 53–60.

[4] R. Behrens, *A grammar based model for XML schema integration*, Proceedings of the 17th British National Conference on Databases: Advances in Databases (BNCOD'00), Exeter, UK, Jul 2000, pp. 172–190.

[5] L. Bellatreche, N. X. Dung, G. Pierra, and D. Hondjack, *Contribution of ontology-based data modeling to automatic integration of electronic catalogues within engineering databases*, Computers in Industry **57** (2006), no. 8-9, 711–724.

[6] P. Buneman, S. Davidson, and A. Kosky, *Theoretical aspects of schema merging*, Proceedings of the 3rd International Conference on Extending Database Technology (EDBT'92), Vienna, Austria, Mar 1992, pp. 152–167.

[7] S. Y. Chien, V. J. Tsotras, C. Zaniolo, and D. Zhang, *Efficient complex query support for multiversion XML documents*, Proceedings of the 8th International Conference on Extending Database Technology (EDBT'02), Prague, Czech Republic, Mar 2002, pp. 161–178.

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

[9] H.-. Do and E. Rahm, *Matching large schemas: Approaches and evaluation*, Information Systems **32** (2007), no. 6, 857–885.

[10] R. M. Duwairi, *Clustering semantically related classes in a heterogeneous multidatabase system*, Information Sciences **162** (2004), no. 3-4.

[11] M. García-Solaco, F. Saltor, and M. Castellanos, *A structure based schema integration methodology*, Proceedings of the 11th International Conference on Data Engineering (ICDE'95), Taipei, Taiwan, Mar 1995, pp. 505–512.

[12] Q. He and T. W. Ling, *An ontology based approach to the integration of entityrelationship schemas*, Data & Knowledge Engineering **58** (2006), no. 3, 299–326.

[13] P. Johannesson, *Using conceptual graph theory to support schema integration*, Proceedings of the 12th International Conference on the Entity-Relationship Approach (ER'93), Arlington, Texas, USA, Dec 1993, pp. 283–296.

[14] Z. Kaoudi, T. Dalamagas, and T. Sellis, *Rdfsculpt: Managing rdf schemas under set-like semantics*, Proceedings of the 2nd European Semantic Web Conference (ESWC'05), Heraklion, Greece, May 2005.

[15] W. Kim, H. T. Chou, and J. Banerjee, *Operations and implementation of complex objects*, Proceedings of the 3rd International Conference on Data Engineering (ICDE'87), Taipei, Taiwan, Feb 1987, pp. 626–633.

[16] J.-L. Koh and A. L. P. Chen, *Integration of heterogeneous object schemas*, Proceedings of the 12th International Conference on the Entity-Relationship Approach (ER'93), Arlington, Texas, USA, Dec 1993, pp. 297–314.

[17] P. S. Mah, , and S. M. Chung, *Schema integration and transaction management for multidatabases*, Information Sciences **111** (1998), no. 1-4.

[18] V. M. Markowitz, *A relation merging technique for relational databases*, Proceedings of the 8th International Conference on Data Engineering (ICDE'92), Tempe, Arizona, Feb 1992, pp. 428–437.

[19] 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.

[20] S. Melnik, E. Rahm, and P. A. Bernstein, *RONDO: a programming platform for generic model management*, Proceedings of the ACM SIGMOD Conference, San Diego, California, Jun 2003, pp. 193–204.

[21] 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.

[22] B. Mitschang, *Extending the relational algebra to capture complex objects*, Proceedings of the 15th International Conference on Very Large Data Bases (VLDB'89), Amsterdam, The Netherlands, Aug 1989, pp. 297–305.

[23] N. Noy, *Semantic integration: A survey of ontology-based approaches*, ACM SIGMOD Record **33** (2004), no. 4.

[24] E. A. Rundensteiner and L. Bic, *Set operations in a data model supporting complex objects*, Proceedings of the International Conference on Extending Database Technology (EDBT'90), Venice, Italy, Mar 1990, pp. 286–300.

[25] I. Schmitt and G. Saake, *Merging Inheritance Hierarchies for Database Integration*, Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (COOPIS'98), New York, USA, 1998, pp. 322–331.

[26] P. Shvaiko and J. Euzenat, *A survey of schema-based matching approaches*, Journal on Data Semantics **IV** (2005).

[27] G. Stumme and A. Maedche, *FCA-MERGE: Bottom-up merging of ontologies*, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'01), Seattle, Washington, USA, Aug 2001, pp. 225–234.

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

[29] H. Xiao and I. F. Cruz, *Integrating and exchanging xml data using ontologies*, Journal on Data Semantics **VI** (2006), 67–89.

[30] S. Yi, B. Huang, and W. T. Chan, *XML application schema matching using similarity measure and relaxation labeling*, Information Sciences **169** (2005), no. 1-2.

[31] H. Zhao and S. Ram, *Combining schema and instance information for integrating heterogeneous data sources*, Data & Knowledge Engineering **61** (2007), no. 2, 281–303.