An Algebra for Specifying Valid Compound Terms in Faceted Taxonomies

Yannis Tzitzikas[1,2], Anastasia Analyti[2], Nicolas Spyratos[3], Panos Constantopoulos[2,4]

[1] *Department of Computer Science, University of Crete, Greece*
[2] *Institute of Computer Science, FORTH-ICS, Greece*
[3] *Laboratoire de Recherche en Informatique, Universite de Paris-Sud, France*
[4] *Department of Informatics, Athens University of Economics and Business, Greece*
Email : tzitzik@ics.forth.gr, analyti@ics.forth.gr, spyratos@lri.fr, panosc@aueb.gr

Abstract

A faceted taxonomy is a set of taxonomies, each describing a given knowledge domain from a different aspect. The indexing of the domain objects is done using compound terms, i.e. conjunctive combinations of terms from the taxonomies. A faceted taxonomy has several advantages over a single taxonomy, including conceptual clarity, compactness, and scalability. A drawback, however, is the cost of identifying compound terms that are invalid, i.e. terms that do not apply to any object of the domain. This need arises both in indexing and retrieval, and involves considerable human effort for specifying the valid compound terms one by one. In this paper, we propose and present in detail an algebra which can be used to specify the set of valid compound terms in an efficient and flexible manner. It works on the basis of the original simple terms of the facets and a small set of positive and/or negative statements. In each algebraic operation, we adopt a closed-world assumption with respect to the declared positive or negative statements. In this paper we elaborate on the properties of the algebraic operators and we describe application and methodological issues.

Keywords: conceptual modelling, faceted taxonomies, semantics, information services on the web.

1 Introduction

There are several application areas where a *taxonomy* is used for indexing the objects of a knowledge domain (e.g. documents, books, product descriptions, Web pages). In recent years taxonomies have become an important factor in organizing knowledge for both the Web and corporate intranets [13, 15, 9]. For instance, Web catalogs, such as Yahoo! or Open Directory, use taxonomies for indexing the pages of the Web. These catalogs turn out to be very useful for browsing and querying the Web. Although they index only a fraction of the pages that are indexed by search engines using statistical methods (e.g. Google), they are hand-crafted by domain experts and are therefore of high quality. Recently, the various search engines have begun to exploit these catalogs in order to enhance the quality of retrieval and also to offer new functionalities. Specifically, search engines now employ catalogs for computing "better" degrees of relevance, and for determining (and presenting to the user) a set of relevant pages for each page in the answer set. In addition, some search engines (e.g. Google) now employ taxonomies in order to enable limiting the scope (or defining the context) of the search. For example, using Google, one can first select a category, e.g. `Sciences/CS/DataStructures`, from the taxonomy of Open Directory and then submit a natural language query, e.g. "`Tree`". The search engine will compute the degree of relevance, with respect to the natural language query, "`Tree`", only of those pages that fall in the category `Sciences/CS/DataStructures` in the catalog of Open Directory. Clearly, this enhances the precision of retrieval and reduces the computational cost (e.g. see [27], [21]).

A taxonomy is a hierarchically-organized set of terms. In designing a taxonomy, one has to define (a priori) appropriate terms and their subterms, according to various criteria. One basic criterion is that each term must be *valid*, in the sense that it applies to, or *indexes*, at least one object of the underlying domain. However, as pointed out long ago [26], the design of a taxonomy can be done in a more convenient and a more systematic manner, if we first identify a number of different aspects of the domain and then design one taxonomy per aspect. This process results in a *faceted taxonomy*, i.e. a *set* of taxonomies, called *facets*.

For example, assume that the domain of interest is a set of hotel Web pages in Greece, and suppose that we want to provide access to these pages according to the *Location* of the hotels and the *Sports* facilities they offer. Figure 1 shows these two facets. Each object can now be described using a *compound term*, i.e. a set of terms containing one or more terms from each facet[1]. For example, a hotel in Crete providing sea ski and wind-surfing facilities would be described by the compound term $\{Crete, SeaSki, Windsurfing\}$.

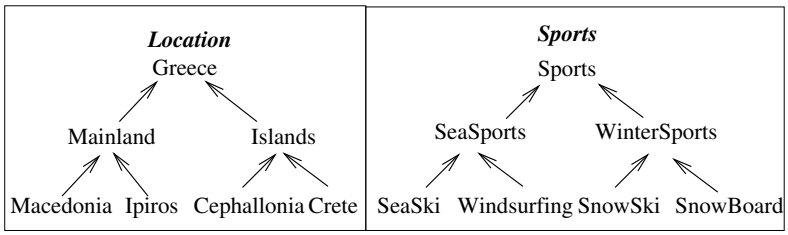

Figure 1: Two facets

The use of faceted taxonomies has several consequences. For example, consider two schemes for describing the objects of a domain, one using a single taxonomy consisting of 50 terms, and the other using a faceted taxonomy consisting of 5 facets each having 10 terms. The first scheme has 50 indexing terms while the second has 5^{10} compound indexing terms! Although both schemes have the same storage requirements, i.e. each one requires storing 50 terms, the second scheme has tremendously more discriminating power than the first.

Overall, a faceted taxonomy has several advantages by comparison to a single taxonomy, such as conceptual clarity, compactness, and scalability (e.g. see [24]). Unfortunately, faceted taxonomies have a serious drawback. Indeed, assuming that each facet has been designed correctly, every single term will be valid. However, even if this assumption holds for every term in every facet, it may not hold for every conceivable compound term. That is, there may exist invalid compound terms, in the sense that there may be no object of the underlying domain indexed simultaneously by all terms present in the compound term. For example, it may very well be that the terms *Crete* (from *Location*) and *SnowBoard* (from *Sports*) are each valid, i.e., there are hotels located in Crete and there are hotels that offer snow-board facilities. However, this does not guarantee that the compound term $\{Crete, SnowBoard\}$ is valid. In fact, there is no hotel in Crete offering snow-board facilities. It follows that not all compound terms of a faceted taxonomy are valid.

Invalid compound terms cause serious problems in indexing and browsing that prevent the design and deployment of faceted taxonomies for real and large scale applications. The ability to infer the valid compound terms of a faceted taxonomy would be very useful. It could be exploited in the indexing process in order to aid the indexer and prevent indexing errors. Such an aid is especially important in cases where the indexing is done by many people who are not domain experts. For example, the indexing of Web pages in the Open Directory (which is used by Netscape, Lycos, HotBot and several other search engines) is done by more than 20.000 volunteer human editors (indexers). On the other hand, the inability to infer valid compound terms may give rise to problems in browsing, as invalid terms will yield no objects. Moreover, if we could infer the valid compound terms in a faceted taxonomy then we would be able to generate navigation trees *on the fly*, having only valid compound terms as nodes.

[1]The interpretation of a compound term is the conjunction of its elements.

The main goal of this paper is to propose an algebra over sets of compound terms, called *compound terminologies*, whose operators allow the efficient and flexible specification of the valid compound terms of the faceted taxonomies. This way, the main drawback of faceted taxonomies, i.e. the formation of invalid compound terms, is alleviated. We refer to this algebra, as the *Compound Term Composition Algebra* (*CTCA*). Following our approach, given a faceted taxonomy, one can use an *algebraic expression* to define the desired set of compound terms. In each algebraic operation, the designer has to declare either a small set of valid compound terms from which other valid compound terms are inferred, or a small set of invalid compound terms from which other invalid compound terms are inferred. By adopting a closed-world assumption, the rest of the compound terms in the compound terminology, that results by combining (the compound terms from) the input compound terminologies, are considered invalid or valid, respectively. This means that the designer can specify the large number of valid compound terms in a faceted taxonomy by providing a relatively small number of (valid or invalid) compound terms. This is an important feature as it minimizes the effort needed by the designer. A distinctive feature of our approach is that there is no need to store the valid compound terms defined by the expression. We only have to store the defining expression as we provide an inference mechanism which can check whether a compound term belongs to the result of an expression. Thus, our method has low storage space requirements.

This paper elaborates in detail the ideas first sketched at [38]. Apart from discussing the properties of each algebraic operator, this paper provides formal proofs and algorithms for all needed tasks, and discusses applicability and methodological issues for using CTCA and for handling evolution.

The remaining of this paper is organized as follows: Section 2 describes formally taxonomies, compound taxonomies, and facets. Section 3 discusses compound term validity and invalidity. Section 4 describes the proposed algebra and Section 5 illustrates its application by an example. Section 6 provides an algorithm for checking whether a compound term belongs to the compound taxonomy defined by an algebraic expression (as well as other algorithms). Section 7 describes a mechanism for generating navigation trees. Section 8 discusses the design process for algebraic expressions and Section 9 discusses implementation issues and applications. Section 10 discusses related work, and finally, Section 11 concludes the paper. Proofs are given in Appendix A.

2 Taxonomies, Compound Taxonomies, and Facets

Def 2.1 A *terminology* is a finite set of names, called *terms*.

Def 2.2 A *taxonomy* is a pair $(\mathcal{T}, \leq)$, where $\mathcal{T}$ is a *terminology* and $\leq$ is a reflexive and transitive relation over $\mathcal{T}$, called *subsumption*.

If a and b are terms of $\mathcal{T}$ and $a \leq b$ then we say that a is *subsumed* by b, or that b *subsumes* a. We also say that a is *narrower* than b, or that b is *broader* than a. For example, `Databases` $\leq$ `Informatics`. We say that two terms a and b are *equivalent*, and write $a \sim b$, if both $a \leq b$ and $b \leq a$ hold, e.g., `Computer Science` $\sim$ `Informatics`. Note that the subsumption relation is a preorder over $\mathcal{T}$ and that $\sim$ is an equivalence relation over the terms of $\mathcal{T}$. Moreover, $\leq$ is a partial order over the equivalence classes of terms. Thus, $(\mathcal{T}, \leq)$ is a poset (up to equivalence).

When using diagrams to depict a taxonomy, such as the ones of Figure 1, term subsumption is indicated by a continuous-line arrow from the narrower term to the broader term. Note that we do not represent the entire subsumption relation but only a subset sufficient for generating it. In particular, we do not represent the reflexive, nor the transitive arrows of the subsumption relation. Equivalence of terms is indicated by a continuous, non-oriented line segment. In what follows, we shall often write $\mathcal{T}$ instead of $(\mathcal{T}, \leq)$, whenever no ambiguity is possible.

We now introduce the concept of compound taxonomy, a basic concept for the rest of this paper. First, we define compound terms over a given taxonomy and their ordering. In all definitions that follow, we assume an underlying taxonomy $(\mathcal{T}, \leq)$.

Def 2.3 A *compound term* over $\mathcal{T}$ is any subset of $\mathcal{T}$.

For example, the following sets of terms are compound terms over the taxonomy *Sports* of Figure 1: $s_1 = \{SeaSki, Windsurfing\}$, $s_2 = \{SeaSports, WinterSports\}$, $s_3 = \{Sports\}$, and $s_4 = \emptyset$.

We denote by $P(\mathcal{T})$ the set of all compound terms over $\mathcal{T}$ (i.e. the powerset of $\mathcal{T}$).

Def 2.4 A *compound terminology* S over $\mathcal{T}$ is any set of compound terms that contains the compound term $\emptyset$.

Clearly, $P(\mathcal{T})$ is a compound terminology over $\mathcal{T}$. The set of all compound terms over $\mathcal{T}$ can be ordered using the following ordering derived from $\leq$, which is actually the *Smyth ordering* on power domains [28].

Def 2.5 Let s, s' be two compound terms over $\mathcal{T}$. The *compound ordering* over $\mathcal{T}$ is defined as follows:
$s \preceq s'$ iff $\forall t' \in s' \ \exists t \in s$ such that $t \leq t'$.

That is, $s \preceq s'$ iff s contains a narrower term for every term of s'. In addition, s may contain terms not present in s'. Roughly, $s \preceq s'$ means that s carries more specific information (or, more discriminating power) than s'. Figure 2.(a) shows the compound ordering of compound terms from Figure 1. Note that $s_1 \preceq s_3$, as s_1 contains *SeaSki* which is a term narrower than the unique term *Sports* of s_3. On the other hand, $s_1 \not\preceq s_2$, as s_1 does not contain a term narrower than *WinterSports*. Finally, $s_2 \preceq s_3$ and $s_3 \preceq \emptyset$. In fact, $\mathcal{T} \preceq s \preceq \emptyset$, for every compound term s.

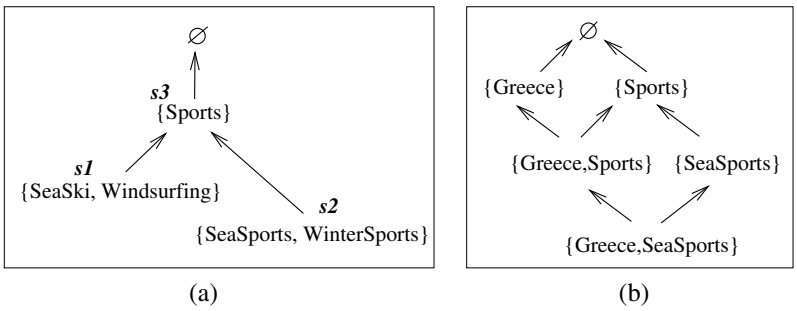

Figure 2: Two examples of compound taxonomies

Also note that while the relation $\leq$ is provided explicitly by the designer, the relation $\preceq$ is *derived* from $\leq$ according to the previous definition.

We say that two compound terms s_1, s_2 are *equivalent*, denoted by $s_1 \sim s_2$ if both $s_1 \preceq s_2$ and $s_2 \preceq s_1$ hold. Clearly, the relation $\sim$ is reflexive, symmetric, and transitive. That is, it is an equivalence relation. The relation $\sim$ partitions $P(\mathcal{T})$ into a set of equivalence classes. Intuitively, equivalent compound terms carry the same information. For example, consider the terminology *Sports* of Figure 1. The compound terms $\{SeaSports, Sports\}$ and $\{SeaSports\}$ are equivalent. From each equivalence class, we consider only one representative compound term.

Def 2.6 A *compound taxonomy* over $\mathcal{T}$ is a pair $(S, \preceq)$, where S is a compound terminology over $\mathcal{T}$, and $\preceq$ is the compound ordering over $\mathcal{T}$ restricted to S.

Figure 2 shows two examples of compound taxonomies. Clearly, in a compound taxonomy $(S, \preceq)$, $\preceq$ is a reflexive and transitive relation over S, and therefore a preorder over S. Additionally, $\preceq$ is antisymmetric (up to equivalence). Therefore, $(S, \preceq)$ is a poset (up to equivalence). Note that, $(P(\mathcal{T}), \preceq)$ is a compound taxonomy over $\mathcal{T}$, $\emptyset$ is the largest element of $P(\mathcal{T})$, and $\mathcal{T}$ is the smallest (up to equivalence). As every subset $\{s_1, s_2\}$ of $P(\mathcal{T})$ has a greatest lower bound (equal to $s_1 \cup s_2$) and a lowest upper bound (equal to $s_1 \cap s_2$), $(P(\mathcal{T}), \preceq)$ is a lattice (up to equivalence).

Let s be a compound term over $\mathcal{T}$. The broader and the narrower compound terms of s are defined as follows: $\mathrm{Br}(s) = \{s' \in P(\mathcal{T}) \mid s \preceq s'\}$ and $\mathrm{Nr}(s) = \{s' \in P(\mathcal{T}) \mid s' \preceq s\}$. In lattice theory terms [6], $\mathrm{Br}(s)$ is the *principal order filter generated by* s and $\mathrm{Nr}(s)$ is the *principal order ideal generated by* s.

Let S be a set of compound terms over $\mathcal{T}$. The broader and the narrower compound terms of S are defined as follows:

$$\begin{aligned} Br(S) &= \cup\{\mathrm{Br}(s) \mid s \in S\} \\ Nr(S) &= \cup\{\mathrm{Nr}(s) \mid s \in S\} \end{aligned}$$

In lattice theory terms, $Br(S)$ is the *order filter generated by* S and $Nr(S)$ is the *order ideal generated by* S.

Def 2.7 A compound terminology S is called an *order filter* (*o-filter*) if $Br(S) = S$. Additionally, S is called an *order ideal* (*o-ideal*) if $Nr(S) = S$.

Def 2.8 The compound terminologies S_i over $\mathcal{T}_i$, $i = 1, ..., n$, are called *separated* if the terminologies $\mathcal{T}_i$, $i = 1, ..., n$, are mutually disjoint.

As we will see in subsection 4.3, the input compound terminologies of the operations of a well-formed (algebraic) expression e are *o-filters* and separated.

As already mentioned, one way of designing a taxonomy is by identifying a number of different aspects of the domain of interest and then designing one taxonomy per aspect. As a result, we obtain a set of taxonomies, called *facets*. Given a set of facets we can define a *faceted taxonomy*, as follows:

Def 2.9 Let $\{F_1, ..., F_k\}$ be a finite set of taxonomies, where $F_i = (\mathcal{T}_i, \leq_i)$, and assume that the terminologies $\mathcal{T}_1$, ..., $\mathcal{T}_k$ are pairwise disjoint. Then, the pair $\mathcal{F} = (\mathcal{T}, \leq)$, where $\mathcal{T} = \bigcup_{i=1}^{k} \mathcal{T}_i$ and $\leq\ = \bigcup_{i=1}^{k} \leq_i$, is a taxonomy which we shall call the *faceted taxonomy generated* by $\{F_1, ..., F_k\}$, or the *family of taxonomies* $\{F_1, ..., F_k\}$. We shall call the taxonomies $F_1, ..., F_k$, the *facets* of $\mathcal{F}$.

It is common practice to refer to a facet through its top term. For example, we refer to the facets of Figure 1 as *Location* and *Sports*. Clearly, all definitions introduced so far apply also to faceted taxonomies. For example, consider the faceted taxonomy $(\mathcal{T}, \leq)$ generated by the two facets of Figure 1. Then, the set $S = \{\{Greece\}, \{Sports\}, \{SeaSports\}, \{Greece, Sports\}, \{Greece, SeaSports\}, \emptyset\}$, is a compound terminology over $\mathcal{T}$. The set S together with the compound ordering of $\mathcal{T}$ (restricted to S) is a compound taxonomy over $\mathcal{T}$. This compound taxonomy is shown in Figure 2.(b). For reasons of brevity, hereafter we shall omit the term $\emptyset$ from the compound terminologies of our examples and figures.

3 Compound Term Validity/Invalidity

Let us now give a model-theoretic interpretation to faceted taxonomies and to compound taxonomies. We conceptualize the world as a set of objects, that is, we assume an arbitrary domain of discourse and a corresponding set of objects Obj. A typical example of such a domain is a set of Web pages. The only constraint that we impose on the set Obj is that it must be a denumerable set. The set of objects described by a term is the *interpretation* of that term, i.e. :

Def 3.1 Given a terminology $\mathcal{T}$, we call *interpretation* of $\mathcal{T}$ over Obj any function $I : \mathcal{T} \rightarrow 2^{Obj}$. Additionally, we say that an object $o \in Obj$ is *indexed by* a term $t \in \mathcal{T}$, according to an interpretation I, if $o \in I(t)$.

Intuitively, the interpretation $I(t)$ of a term t is the set of objects to which the term t is correctly applied. In our discussion, the set Obj will be usually understood from the context. So, we shall often say simply "an interpretation" instead of "an interpretation over Obj". Interpretation, as defined above, assigns to a term denotational or extensional meaning ([45]).

Now, any interpretation I of $\mathcal{T}$ can be extended to an interpretation $\hat{I}$ of $P(\mathcal{T})$ as follows:

$$\hat{I}(\{t_1, ..., t_n\}) = I(t_1) \cap I(t_2) \cap ... \cap I(t_n)$$

Def 3.2 Let $(\mathcal{T}, \leq)$ be a taxonomy. An interpretation I of $\mathcal{T}$ is a *model* of $(\mathcal{T}, \leq)$, if for each $t, t' \in \mathcal{T}$: if $t \leq t'$ then $I(t) \subseteq I(t')$.

Prop. 3.1 Let S be a compound terminology over a terminology $\mathcal{T}$, and let s and s' be two elements of S. It holds: $s \preceq s'$ iff $\hat{I}(s) \subseteq \hat{I}(s')$ in every model I of $(\mathcal{T}, \leq)$.

We can see that the compound ordering $\preceq$, as defined in Def. 2.5, is also justified semantically (it coincides with the extensional subsumption).

Def 3.3 Let $(\mathcal{T}, \leq)$ be a taxonomy. An interpretation $\hat{I}$ of $P(\mathcal{T})$ is a *model* of $(P(\mathcal{T}), \preceq)$, if for each $s, s' \in P(\mathcal{T})$: if $s \preceq s'$ then $\hat{I}(s) \subseteq \hat{I}(s')$.

From the above, it easily follows that an interpretation I is a model of $(\mathcal{T}, \leq)$ iff $\hat{I}$ is a model of $(P(\mathcal{T}), \preceq)$.

The *current state of affairs* actually restricts the models $\hat{I}$ of $(P(\mathcal{T}), \preceq)$ to the set $\mathcal{M}$ such that $\forall s \in P(\mathcal{T})$, either (i) $\forall \hat{I} \in \mathcal{M}$, $\hat{I}(s) \neq 0$ or (ii) $\forall \hat{I} \in \mathcal{M}$, $\hat{I}(s) = 0$. This means that in the current state of affairs, either there is an object in Obj indexed by all terms in s or there is no object in Obj indexed by s.

We characterize a compound term s over $\mathcal{T}$ as *invalid*, according to the current state of affairs, if for all models $\hat{I} \in \mathcal{M}$, it holds $\hat{I}(s) = \emptyset$. Intuitively, a compound term s is *invalid*, if in the current state of affairs, there is no object in Obj that is indexed by all terms in s. However, the empty extension of a compound term s in a particular materialized faceted taxonomy M (i.e. in a faceted taxonomy accompanied by a stored interpretation I) does not imply that s is an invalid compound term, as it may exist an object in the real world that is indexed by s but this knowledge is not depicted in M. Thus, invalidity implies empty assertional knowledge but not the other way around. Reversely, we characterize a compound term s over $\mathcal{T}$ as *valid*, according to the current state of affairs, if for all models $\hat{I} \in \mathcal{M}$, it holds $\hat{I}(s) \neq \emptyset$. Intuitively, a compound term s is *valid* if in the current state of affairs, there is an object in Obj that is indexed by all terms in s. Note that it is possible for the extension of a valid compound term in a particular materialized faceted taxonomy to be empty. Thus, non-empty assertional knowledge implies validity but not the other way around.

We assume that every singleton compound term is valid. It is easy to see that $P(\mathcal{T})$ is the disjoint union of the sets of valid and invalid compound terms. Moreover, if s is a valid compound term then all compound terms in $\mathrm{Br}(s)$ are valid. Thus, the set of valid compound terms is an *o-filter*. Additionally, if s is an invalid compound term then all compound terms in $\mathrm{Nr}(s)$ are invalid. Thus, the set of invalid compound terms is an *o-ideal*.

The algebraic expressions of the Compound Term Composition Algebra (CTCA), that are introduced in the next section, allow to define the valid and invalid compound terms of a faceted taxonony $(\mathcal{T}, \leq)$, according to the current state of affairs. The semantics of CTCA are formally defined in [37].

4 The Compound Term Composition Algebra

Let $\mathcal{F} = (\mathcal{T}, \leq)$ be the faceted taxonomy generated by a given set of facets $\{F_1, ..., F_k\}$. The problem is that $\mathcal{F}$ does not itself specify which compound terms, i.e. which elements of $P(\mathcal{T})$, are valid and which are not. To tackle this problem, we introduce a method for defining a compound terminology over $\mathcal{T}$ (i.e. a subset of $P(\mathcal{T})$) which consists of those compound terms that the designer considers as valid. The main tool for accomplishing this task is an algebra that we now define. To begin with we associate every facet F_i with a compound terminology T_i that we call the *basic compound terminology* of F_i. The basic compound terminologies are the "building blocks" of the algebra.

Def 4.1 Let $F_i = (\mathcal{T}_i, \leq)$ be a facet. The *basic compound terminology* of F_i is defined as follows:

$$T_i = Br(\{\{t\} \mid t \in \mathcal{T}_i\})$$

As every singleton compound term $\{t\}$ over $\mathcal{T}_i$ is considered valid, all compound terms in $\mathrm{Br}(\{t\})$ are valid. Thus, T_i is the set of compound terms over $\mathcal{T}_i$ that are initially known to be valid. For example, Figure 3 shows a facet, called *Season*, which classifies hotels according to the season they are open. Note that $Allyear \leq Summer$ and $Allyear \leq Winter$. The compound term $\{Summer, Winter\}$ is a valid compound term as there are hotels indexed by *Allyear*, and thus indexed by both *Summer* and *Winter*. Indeed, the basic compound taxonomy of *Season* is: $\{\{Season\}, \{Summer\}, \{Winter\}, \{Allyear\}, \{Summer, Winter\}\}$[2]. In the case that, the terminology $\mathcal{T}_i$ does not contain terms which are subsumed by different terms not related by the subsumption relation (single inheritance), the basic compound terminology of $\mathcal{T}_i$ is simplified as follows: $T_i = \{\ \{t\} \mid t \in \mathcal{T}_i\} \cup \{\emptyset\}$.

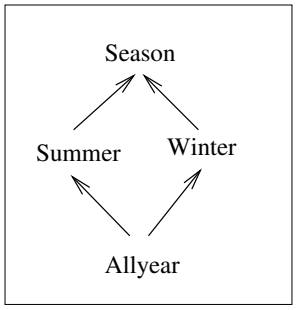

Figure 3: The facet *Season*

Let $\mathcal{S}$ denote the set of all compound terminologies over $\mathcal{T}$. We define an algebra over $\mathcal{S}$, called *Compound Term Composition Algebra* (*CTCA*), which includes four operations and compound terminologies as operands. Compound terms can be formed by combining terms from different facets, but also terms from the same facet. A binary product operation and a unary self-product operation are defined to generate term combinations, respectively. Since not all term combinations are valid, the issue is how to employ available domain knowledge in order to specify only valid compound terms. Such knowledge may be available in positive or negative form: combinations known to be valid or invalid. The issue is dealt by defining four more general operations that include positive or negative modifiers, which are sets of known valid or known invalid compound terms. The unmodified product and self-product operations turn out to be special cases with the modifiers at certain extreme values. Thus, the four operations of the algebra are: *plus-product, minus-product, plus-self-product*, and *minus-self-product*.

For defining the desired compound taxonomy the designer has to formulate an algebraic expression e, using these operations and initial operands the basic compound terminologies $\{T_1, ..., T_k\}$.

Before we describe each operation in detail, we define the auxiliary binary operation $\oplus$ over $\mathcal{S}$, i.e. $\oplus : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$, called *product*.

Def 4.2 Let S and S' be two compound terminologies ($S, S' \in \mathcal{S}$). The *product* of S and S', denoted by $S \oplus S'$, is defined as $S \oplus S' = \{s \cup s' \mid s \in S,\ s' \in S'\}$.

This operation results in an "unqualified" compound terminology whose compound terms are all possible unions of compound terms from its arguments. The compound terms of the result are ordered according to the compound ordering (see Def. 2.5). For example, consider the compound terminologies $S = \{\{Greece\}, \{Islands\}\}$ and $S' = \{\{Sports\}, \{SeaSports\}\}$. The compound taxonomy corresponding to $S \oplus S'$ is shown in Figure 4, and consists of 8 terms.

Recall that for reasons of brevity, we omit the term $\emptyset$ from the compound terminologies of our examples ($\emptyset$ is an element of S, S', and $S \oplus S'$). It can be easily seen that the product operation is commutative and associative, and that it can be easily extended to an n-ary operation: $S_1 \oplus ... \oplus S_n = \{\ s_1 \cup ... \cup s_n \mid s_i \in S_i\}$. Additionally, as $\emptyset \in S_i$, for $i = 1, ..., n$, it holds that $S_i \subseteq S_1 \oplus ... \oplus S_n$. Below we describe each operation of our algebra in detail.

[2]Note that from each equivalent class of compound terms, we consider only a representative compound term.

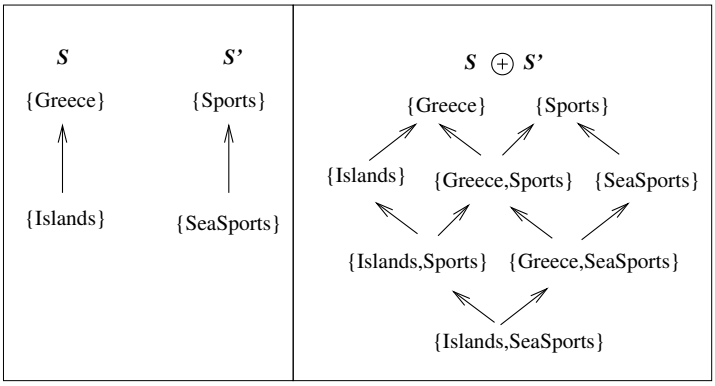

Figure 4: An example of a product $\oplus$ operation

4.1 The *plus-product* and the *minus-product* operations

Consider the compound terminologies S and S' shown in the upper part of Figure 5, and suppose that we want to define a compound terminology that does not contain the compound terms $\{Islands, WinterSports\}$ and $\{Islands, SnowSki\}$, because they are invalid. For this purpose we introduce two "generalizations" of the $\oplus$ operation, namely the *plus-product* and the *minus-product*. Each of these two operations has an extra parameter denoted by P or N, respectively. The set P is a set of compound terms that we certainly want to appear in the result of the operation, i.e. they are valid. On the other hand, the set N is a set of compound terms that we certainly do not want to appear in the result of the operation, i.e. they are invalid. These parameters are declared by domain experts that perform the indexing and allow to infer all compound terms that are valid or invalid.

To proceed we need to distinguish what we shall call *genuine compound terms*. Intuitively, a genuine compound term combines non-empty compound terms from more than one compound terminologies.

Def 4.3 The set of *genuine* compound terms over a set of compound terminologies $S_1, ..., S_n$, denoted by $G_{S_1,...,S_n}$, is defined as follows:

$$G_{S_1,...,S_n} = S_1 \oplus ... \oplus S_n - \bigcup_{i=1}^{n} S_i$$

For example if $S_1 = \{\{Greece\}, \{Islands\}\}$, $S_2 = \{\{Sports\}, \{WinterSports\}\}$, and $S_3 = \{\{Pensions\}, \{Hotels\}\}$ then

$$\begin{aligned} \{Greece, WinterSports, Hotels\} &\in G_{S_1,S_2,S_3}, \\ \{WinterSports, Hotels\} &\in G_{S_1,S_2,S_3}, \text{ but} \\ \{Hotels\} &\notin G_{S_1,S_2,S_3} \end{aligned}$$

Assume that the compound terms of $S_1, ..., S_n$ are valid. We are interested in characterizing the validity of all combinations of compound terms of $S_1, ..., S_n$. As we already know the validity of the compound terms of $S_1, ..., S_n$, we are basically interested in characterizing the validity of the compound terms in $G_{S_1,...,S_n}$. This is done through the following operations, plus-product and minus-product.

We can now define the *plus-product* operation, $\oplus_P$, an n-ary operation over $\mathcal{S}$ ($\oplus_P : \mathcal{S} \times ... \times \mathcal{S} \rightarrow \mathcal{S}$), where the parameter P is a set of valid compound terms from the product of the input compound terminologies. Specifically, the set P is a subset of $G_{S_1,...,S_n}$ (i.e., $P \subseteq G_{S_1,...,S_n}$), as we assume that all compound terms in the operands $S_1, ..., S_n$ are valid.

Def 4.4 Let $S_1, ..., S_n$ be compound terminologies and $P \subseteq G_{S_1,...,S_n}$. The *plus-product* of $S_1, ..., S_n$ with respect to P, denoted by $\oplus_P(S_1, ..., S_n)$, is defined as follows:

$$\oplus_P(S_1, ...S_n) = S_1 \cup ... \cup S_n \ \cup \ Br(P)$$

This operation results in a compound terminology consisting of the compound terms of the initial compound terminologies, *plus* the compound terms which are broader than an element of P. This is because, all compound terms in $S_1 \cup ... \cup S_n \cup Br(P)$ are valid. By adopting a closed-world assumption, all compound terms in $S_1 \oplus ... \oplus S_n - \oplus_P(S_1, ...S_n) = G_{S_1,...,S_n} - Br(P)$ are invalid.

For example, consider the compound terminologies S and S' of Figure 5 and suppose that $P = \{\{Islands, Seasports\}, \{Greece, SnowSki\}\}$. The compound taxonomy of the operation $\oplus_P(S, S')$ is shown in Figure 5. In this figure, we enclose in squares the elements of P. We see that the compound terminology $\oplus_P(S, S')$ contains the compound term $s = \{Greece, Sports\}$, as $s \in$Br$(\{Islands, SeaSports\})$. However, it does not contain the compound terms $\{Islands, WinterSports\}$ and $\{Islands, SnowSki\}$, as they do not belong to $S \cup S' \cup Br(P)$.

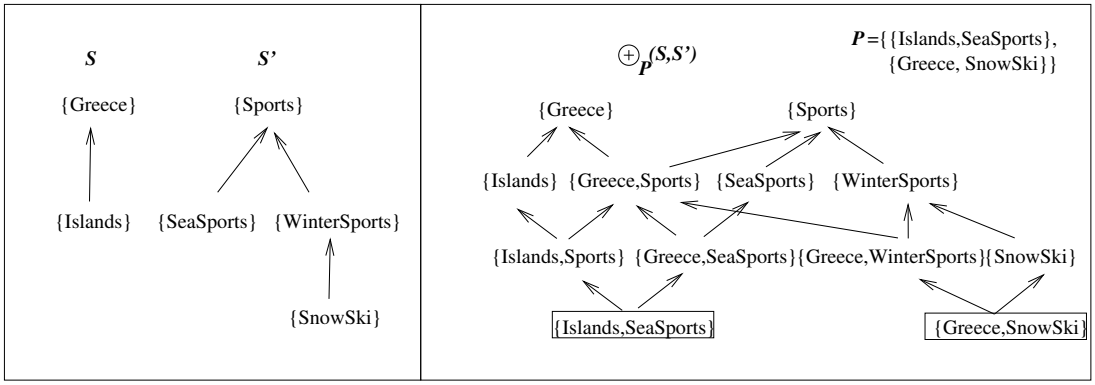

Figure 5: An example of a *plus-product*, $\oplus_P$, operation

The following proposition gives two simplifications of the operation $\oplus_P$ for the two extreme values of the P parameter. The first property says that the product is a special case of the plus-product, while the second property says that if $P = \emptyset$ then the plus-product operation defines a compound terminology that contains only the compound terms of the operands. The third property shows that the plus-product of *o-filters* is an *o-filter*. Finally, the fourth property shows that the operation $\oplus_P$ is commutative.

Prop. 4.1 Let the compound terminologies S_i, $i = 1, ..., n$, and let $P \subseteq G_{S_1,...,S_n}$. It holds:

1. if S_i, $i = 1, ..., n$, are *o-filters* and $P = G_{S_1,...,S_n}$ then $\oplus_P(S_1, ..., S_n) = S_1 \oplus ... \oplus S_n$,
2. if $P = \emptyset$ then $\oplus_P(S_1, ..., S_n) = \bigcup_{i=1}^{n} S_i$,
3. if S_i, $i = 1, ..., n$, are *o-filters* then $\oplus_P(S_1, ..., S_n)$ is an *o-filter*, and
4. $\oplus_P(S_1, ..., S_n) = \oplus_P(S_{l_1}, ..., S_{l_n})$, where $(l_1, ..., l_n)$ is a permutation of $\{1, ..., n\}$. (*Commutativity*)

The following proposition shows that the application of a binary $\oplus_P$ operation on one or two n-ary $\oplus_P$ operations results in a single n-ary $\oplus_P$ operation. Similarly, the application of an n-ary $\oplus_P$ operation on other n-ary $\oplus_P$ operations results in a single n-ary $\oplus_P$ operation (by taking the union of all P parameters in the input).

Prop. 4.2 Let the compound terminologies S_i, for $i = 1, 2, ..., n$, and let $l \leq n - 2$. It holds:

1. $(\oplus_{P1}(S_1, ..., S_{n-1}) \oplus_{P_2} S_n = \oplus_{P_1 \cup P_2}(S_1, ..., S_n)$.
2. $(\oplus_{P_1}(S_1, ..., S_l)) \ \oplus_{P_2} \ (\oplus_{P_3}(S_{l+1}, ..., S_n)) = \oplus_{P_1 \cup P_2 \cup P_3}(S_1, ..., S_n)$.

The following property says that the smaller is the parameter P, the smaller is the resulting compound terminology.

Prop. 4.3 If $P \subseteq P'$ then $\oplus_P(S_1, ..., S_n) \subseteq \oplus_{P'}(S_1, ..., S_n)$

From the previous proposition and Prop. 4.1, it follows that if the compound terminologies S_i, $i = 1, ..., n$, are *o-filters* and $P \subseteq G_{S_1,...,S_n}$ then:

$$\bigcup_{i=1}^{n} S_i \subseteq \oplus_P(S_1, ..., S_n) \subseteq S_1 \oplus ... \oplus S_n$$

The following proposition can be used for optimization, i.e. for minimizing the space needed for storing the parameter P.

Prop. 4.4 $\oplus_P(S_1, ..., S_n) = \oplus_{P'}(S_1, ..., S_n)$ iff $minimal_{\preceq}(P) = minimal_{\preceq}(P')$.

The next proposition shows that the binary $\oplus_P$ operation is associative with appropriate transformations of the P parameters, when the input compound terminologies are *o-filters*.

Prop. 4.5 Let the compound terminologies S_1, S_2, and S_3 be *o-filters*. It holds:

$$\begin{aligned}
&(S_1 \oplus_{P_1} S_2) \oplus_{P_2} S_3 = S_1 \oplus_{P_1'} (S_2 \oplus_{P_2'} S_3), \text{ where} \\
&P_1' = P_1 \cup (P_2 - S_2 \oplus S_3), \\
&P_2' = \{s_2 \cup s_3 \mid \exists s \in P_2,\ \emptyset \neq s_2, s_3 \subseteq s,\ s_2 \in S_2, s_3 \in S_3\}.
\end{aligned}$$

Now we define the *minus-product* operation, $\ominus_N$, an n-ary operation over $\mathcal{S}$ ($\ominus_N : \mathcal{S} \times ... \times \mathcal{S} \rightarrow \mathcal{S}$), where the parameter N is a set of invalid compound terms from the product of the input compound terminologies. Specifically, the set N is a subset of $G_{S_1,...,S_n}$ (i.e., $N \subseteq G_{S_1,...,S_n}$), as we assume that all compound terms in the operands $S_1, ...S_n$ are valid.

Def 4.5 Let $S_1, ..., S_n$ be compound taxonomies and $N \subseteq G_{S_1,...,S_n}$. The *minus-product* of $S_1, ..., S_n$ with respect to N, denoted by $\ominus_N(S_1, ..., S_n)$, is defined as follows:

$$\ominus_N(S_1, ...S_n) = S_1 \oplus ... \oplus S_n - Nr(N)$$

This operation results in a compound terminology consisting of all compound terms in the product of the initial compound terminologies, *minus* all compound terms which are narrower than an element of N. This is because, all compound terms in $Nr(N)$ are invalid. By adopting a closed-world assumption, all compound terms in $\ominus_N(S_1, ...S_n) = S_1 \oplus ... \oplus S_n - Nr(N)$ are valid.

For example, consider the compound terminologies S and S' of the previous example and suppose that $N = \{\{Islands, WinterSports\}\}$. The result of the operation $\ominus_N(S, S')$ is shown in Figure 6. We see that the compound terminology $\ominus_N(S, S')$ does not contain the compound terms $\{Islands, WinterSports\}$ and $\{Islands, SnowSki\}$, as they are elements of $Nr(N)$. Notice that the compound taxonomies of Figures 5 and 6 coincide. These examples demonstrate two alternative ways of defining the desired compound taxonomy.

The following proposition gives two simplifications of the operation $\ominus_N$ for the two extreme values of the N parameter. Note that these two simplifications are the opposite of these of the $\oplus_P$ operation, given in Prop. 4.1. The first property says that if N equals the set of all genuine compound terms then the minus-product contains only the compound terms of the operands. The second property says that the product is a special case of the minus-product. The third property shows that the minus-product of *o-filters* is an *o-filter*. Finally, the fourth property shows that $\ominus_N$ is commutative.

Prop. 4.6 Let the compound terminologies S_i, $i = 1, ..., n$, and let $N \subseteq G_{S_1,...,S_n}$. It holds:

1. if S_i, $i = 1, ..., n$, are separated and $N = G_{S_1,...,S_n}$ then $\ominus_N(S_1, ..., S_n) = \bigcup_{i=1}^{n} S_i$,
2. if $N = \emptyset$ then $\ominus_N(S_1, ..., S_n) = S_1 \oplus ... \oplus S_n$,

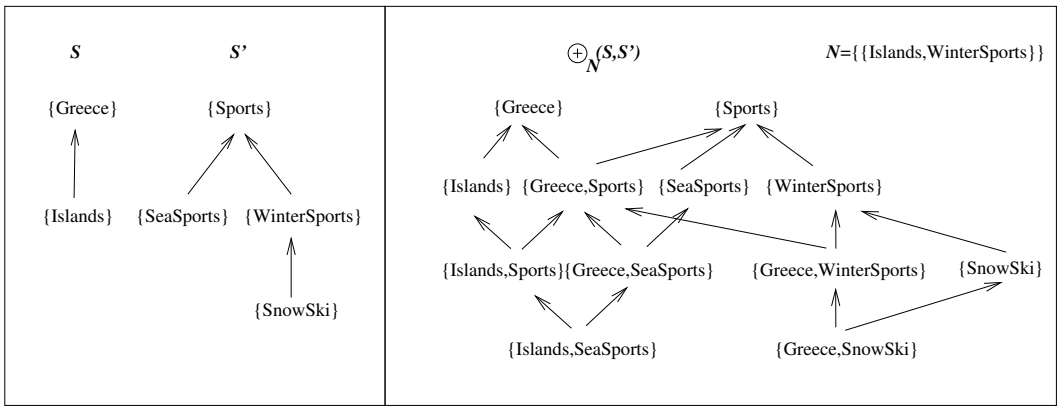

Figure 6: An example of a *minus-product*, $\ominus_N$, operation

3. if S_i, $i = 1, ..., n$, are *o-filters* then $\ominus_N(S_1, ..., S_n)$ is an *o-filter*, and
4. $\ominus_N(S_1, ..., S_n) = \ominus_N(S_{l_1}, ..., S_{l_n})$, where $(l_1, ..., l_n)$ is a permutation of $\{1, ..., n\}$ (*Commutativity*)

The following proposition shows that the application of a binary $\ominus_N$ operation on other n-ary $\ominus_N$ operations results in a single n-ary $\ominus_N$ operation. Similarly, the application of an n-ary $\ominus_N$ operation on other n-ary $\ominus_N$ operations results in a single n-ary $\ominus_N$ operation (by taking the union of all N parameters in the input).

Prop. 4.7 Let the compound terminologies S_i, for $i = 1, ..., n$, be separated and let $l \leq n - 2$. It holds:

1. $\ominus_{N_1}(S_1, ..., S_{n-1}) \ominus_{N_2} S_n = \ominus_{N_1 \cup N_2}(S_1, ..., S_n)$.
2. $(\ominus_{N_1}(S_1, ..., S_l)) \ominus_{N_2} (\ominus_{N_3}(S_{l+1}, ..., S_n)) = \ominus_{N_1 \cup N_2 \cup N_3}(S_1, ..., S_n)$.

The following proposition shows that the smaller is the parameter N, the larger is the resulting compound taxonomy.

Prop. 4.8 If $N \subseteq N'$ then $\ominus_N(S_1, ..., S_n) \supseteq \ominus_{N'}(S_1, ..., S_n)$.

From the above proposition and Prop. 4.6, it follows that if the compound terminologies S_i, $i = 1, ..., n$, are separated and $N \subseteq G_{S_1,...,S_n}$ then:

$$\bigcup_{i=1}^{n} S_i \subseteq \ominus_N(S_1, ..., S_n) \subseteq S_1 \oplus ... \oplus S_n$$

The following proposition can be used for optimization, i.e. for minimizing the space needed for storing the parameter N.

Prop. 4.9 $\ominus_N(S_1, ..., S_n) = \ominus_{N'}(S_1, ..., S_n)$ iff $maximal_{\preceq}(N) = maximal_{\preceq}(N')$.

The next proposition shows that the binary $\ominus_N$ operation is associative with appropriate transformations of the N parameters, when the input compound terminologies are separated.

Prop. 4.10 Let the compound terminologies S_1, S_2, S_3 be separated. It holds:

$$(S_1 \ominus_{N_1} S_2) \ominus_{N_2} S_3 = S_1 \ominus_{N_1'} (S_2 \ominus_{N_2'} S_3), \text{ where}$$
$$N_1' = N_1 \cup (N_2 - S_2 \oplus S_3),$$
$$N_2' = N_2 \cap (S_2 \oplus S_3).$$

4.2 The *Self-product* operations

The operators introduced so far allow defining a compound terminology which consists of compound terms that contain at most one compound term from each basic compound terminology. However, a valid compound term may contain any set of terms of the same facet (multiple classification). To capture such cases, we define the *self-product*, $\overset{*}{\oplus}$, a unary operation which gives all possible compound terms of one facet. Subsequently, we shall modify this operation with the parameters P and N.

Let $\mathcal{BS}$ be the set of basic compound terminologies, that is $\mathcal{BS}= \{T_1, ..., T_k\}$.

Def 4.6 Let T_i be a basic compound terminology. The *self-product* of T_i, denoted by $\overset{*}{\oplus}(T_i)$, is defined as: $\overset{*}{\oplus}(T_i) = P(\mathcal{T}_i)$.

For example, consider the facet *Sports* of Figure 1. The compound terms $\{SeaSports, WinterSports\}$ and $\{SeaSki, Windsurfing, WinterSports\}$ are elements of $\overset{*}{\oplus}(Sports)$.

The notion of genuine compound terms is also necessary here.

Def 4.7 The set of *genuine* compound terms over a basic compound terminology T_i, denoted by G_{T_i}, is defined as follows: $G_{T_i} = \overset{*}{\oplus}(T_i) - T_i$

Now we define the *plus-self-product* operation, $\overset{*}{\oplus}_P$, a unary operation ($\overset{*}{\oplus}_P$: $\mathcal{BS} \rightarrow \mathcal{S}$) where the parameter P is a set of compound terms that we want to appear in the result of the operation. The set P is a subset of G_{T_i}.

Def 4.8 Let T_i be a basic compound terminology and $P \subseteq G_{T_i}$. The *plus-self-product* of T_i with respect to P, denoted by $\overset{*}{\oplus}_P(T_i)$, is defined as follows:

$$\overset{*}{\oplus}_P(T_i) = T_i \cup Br(P)$$

This operation results in a compound terminology consisting of the compound terms of the initial basic compound terminology, *plus* all compound terms which are broader than an element of P. For example, the result of the operation $\overset{*}{\oplus}_P(Sports)$, where $P = \{\{SeaSki, Windsurfing\}, \{SnowSki, SkiBoard\}\}$ is shown in Figure 7. It is easy to see that the analogous of Prop. 4.1 with regard to the extreme cases of P holds, namely,

$$\overset{*}{\oplus}_{G_{T_i}}(T_i) = \overset{*}{\oplus} T_i \quad \text{and} \quad \overset{*}{\oplus}_{\emptyset} = T_i$$

Additionally, for any parameter P it holds: $T_i \subseteq \overset{*}{\oplus}_P(T_i) \subseteq \overset{*}{\oplus}(T_i)$

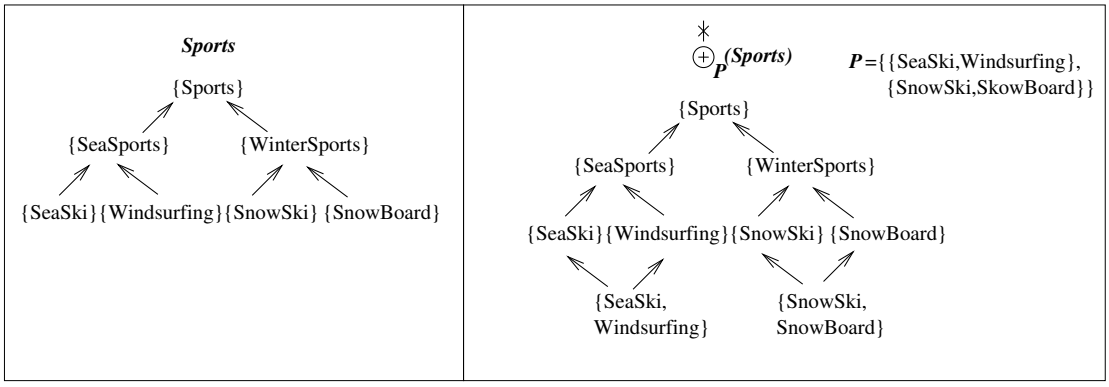

Figure 7: An example of a *plus-self-product*, $\overset{*}{\oplus}_P$, operation

The following definition introduces the *minus-self-product* operation, $\overset{*}{\ominus}_N$, a unary operation ($\overset{*}{\ominus}_N$: $\mathcal{BS} \rightarrow \mathcal{S}$) where the parameter N is a set of compound terms that we do not want to appear in the result of the operation. The set N is a subset of G_{T_i}.

Def 4.9 Let T_i be a basic compound terminology and $N \subseteq G_{T_i}$. The *minus-self-product* of T_i with respect to N, denoted by $\overset{*}{\ominus}_N(T_i)$, is defined as follows:

$$\overset{*}{\ominus}_N(T_i) = \overset{*}{\oplus}(T_i) - Nr(N)$$

This operation results in a compound terminology consisting of all compound terms in the self-product of T_i, *minus* the compound terms which are narrower than an element in N. For example, we can obtain the compound terminology of Figure 7 by the operation $\overset{*}{\ominus}_N(Sports)$, where $N = \{\{SeaSports, WinterSports\}\}$. Concerning the extreme cases of N, it is easy to see that the analogous of Prop. 4.6 holds:

$$\overset{*}{\ominus}_{G_{T_i}}(T_i) = T_i \text{ and } \overset{*}{\ominus}_{\emptyset}(T_i) = \overset{*}{\oplus} T_i$$

Additionally, for any parameter N it holds: $T_i \subseteq \overset{*}{\ominus}_N(T_i) \subseteq \overset{*}{\oplus} T_i$

Prop. 4.11 Let $\mathcal{T}_i$ be a basic compound terminology and $P, N \subseteq G_{T_i}$. Then, the compound terminologies T_i, $\overset{*}{\oplus}_P(T_i)$, and $\overset{*}{\ominus}_N(T_i)$ are *o-filters*.

4.3 Algebraic Expressions

For defining the desired compound taxonomy, the designer has to formulate an expression e, where an expression is defined as follows:

Def 4.10 An expression over a faceted taxonomy $(\mathcal{T}, \leq)$ is defined according to the following grammar:

$$e ::= \oplus_P(e, ..., e) \mid \ominus_N(e, ..., e) \mid \overset{*}{\oplus}_P T_i \mid \overset{*}{\ominus}_N T_i \mid T_i$$

The outcome of the evaluation of an expression e is denoted by S_e and is called the *compound terminology* of e. In addition, $(S_e, \preceq)$ is called the *compound taxonomy* of e. All compound terms in S_e are *valid* and the rest in $P(\mathcal{T}_e) - S_e$ are *invalid*, where $\mathcal{T}_e$ is the union of the terminologies of the facets appearing in e.

We are interested only in *well-formed* expressions, defined as follows:

Def 4.11 An expression e is *well-formed* iff:

(i) each basic compound terminology T_i appears at most once in e,

(ii) each parameter P that appears in e, is a subset of the associated set of genuine compound terms, e.g. if $e = \oplus_P(e_1, e_2)$ or $e = \overset{*}{\oplus}_P(T_i)$ then it should be $P \subseteq G_{S_{e_1}, S_{e_2}}$ or $P \subseteq G_{T_i}$, respectively, and

(iii) each parameter N that appears in e, is also a subset of the associated set of genuine compound terms, e.g. if $e = \ominus_N(e_1, e_2)$ or $e = \overset{*}{\ominus}_N(T_i)$ then it should be $N \subseteq G_{S_{e_1}, S_{e_2}}$ or $N \subseteq G_{T_i}$, respectively.

For example, the expression $(T_1 \oplus_P T_2) \ominus_N T_1$ is not well-formed, as T_1 appears twice in the expression.

Constraints (i), (ii), and (iii) ensure that we have *no conflicts*, meaning that that the valid and invalid compound terms of an expression e increase as the length of e increases. For example, if we omit constraint (i) then an invalid compound term according to an expression $T_1 \oplus_P T_2$ could be valid according to a larger expression $(T_1 \oplus_{P_1} T_2) \oplus_{P_2} T_1$. If we omit constraint (ii) then an invalid compound term according to an expression $T_1 \oplus_{P_1} T_2$ could be valid according to a larger expression $(T_1 \oplus_{P_1} T_2) \oplus_{P_2} T_3$. Additionally, if we omit constraint (iii) then a valid compound term according to an expression $T_1 \oplus_P T_2$ could be invalid according to a larger expression $(T_1 \oplus_P T_2) \ominus_N T_3$. This monotonic behavior in the evaluation of a well-formed expression results in a number of useful

properties. Specifically, due to their monotonicity, well-formed expressions can be formulated in a systematic, gradual manner (intermediate results of subexpressions are not invalidated by larger expressions). This will become more clear in the example of Section 5.

We would like to note that constraint (ii) can be relaxed to the following: (ii)′ each parameter P that appears in e, is a subset of the product of the input compound terminologies. However, we prefer constraint (ii) for uniform and symmetrical presentation of P and N.

A direct consequence of Propositions 4.1, 4.6, and 4.11 is that the compound terminology S_e of any well-formed expression e (and any subexpression of e) is an *o-filter*. Additionally, due to property (i) of a well-formed expression, the input compound terminologies S_{e_1}, ..., S_{e_n} of any subexpression $\oplus_P(e_1, ..., e_n)$ or $\ominus_N(e_1, ..., e_n)$ of a well-formed expression e are separated.

We would like to note that our theory does not depend on a strict definition of validity. It is up to the designer to decide which compound terms $V \subseteq P(\mathcal{T})$ should be considered valid in a particular application. The only constraint is that V should be an *o-filter*, that is if a compound term s is in V then all compound terms broader than s are also in V. Indeed, in [36], we show that for any compound terminology V that is an *o-filter*, there is a well-formed expression e of any expression form[3] such that $S_e = V$.

In the rest of the paper, *we assume that expressions are well-formed.* The algorithm that checks if an expression is well-formed is given in subsection 6.2.

5 Example

Suppose that the domain of interest is a set of hotel Web pages and that we want to index these pages using a faceted taxonomy. First, we must define the taxonomy. Suppose it is decided to do the indexing according to three facets, namely the *location* of the hotels, the kind of *accommodation*, and the *facilities* they offer. Specifically, assume that the designer employs (or designs from scratch) the facets shown in Figure 8. This faceted taxonomy has 13 terms ($|\mathcal{T}|$=13) and $P(\mathcal{T})$ has 890 compound terms[4].

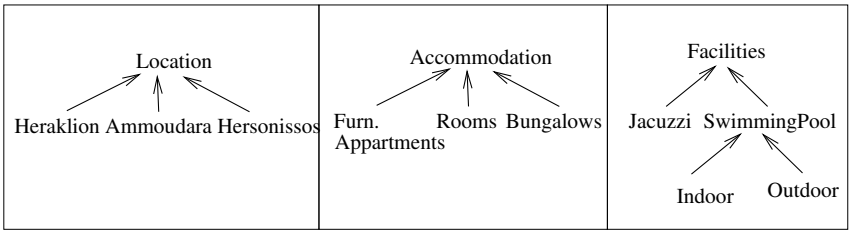

Figure 8: Three-facets

However, available domain knowledge suggests that only 96 compound terms are valid. Omitting the compound terms which are singletons or contain top terms of the facets, the following 23 valid compound terms remain:

$\{Heraklion, Furn.Appartments, \}, \{Heraklion, Rooms\}, \{Ammoudara, Furn.Appartments\},$
$\{Ammoudara, Rooms\}, \{Ammoudara, Bungalows\}, \{Hersonissos, Furn.Appartments\},$
$\{Hersonissos, Rooms\}, \{Hersonissos, Bungalows\}, \{Hersonissos, SwimmingPool\},$
$\{Hersonissos, Indoor\}, \{Hersonissos, Outdoor\}, \{Ammoudara, Jacuzzi\},$
$\{Rooms, SwimmingPool\}, \{Rooms, Indoor\}, \{Bungalows, SwimmingPool\},$
$\{Bungalows, Outdoor\}, \{Bungalows, Jacuzzi\}, \{Hersonissos, Rooms, SwimmingPool\},$

[3]We use the term *expression form* to refer to an algebraic expression whose P and N parameters are indefined (unspecified). Note that an expression form can be represented as a parse tree.

[4]Recall that equivalent compound terms are considered only once. Thus, $|P(\mathcal{T})|$ is not 2^{13} but 890. This is computed as follows: It holds that $|\overset{*}{\oplus}(Location)|$=8, $|\overset{*}{\oplus}(Accomodation)| = 8$, and $|\overset{*}{\oplus}(Facilities)| = 10$. Thus, $|P(\mathcal{T})| = |(\overset{*}{\oplus}(Location)) \oplus (\overset{*}{\oplus}(Accomodation)) \oplus (\overset{*}{\oplus}(Facilities)|| = (8 + 8 * 8 + 8 * 10 + 8 * 8 * 10) + (8 + 8 * 10) + 10 = 890$.

$\{Hersonissos, Rooms, Indoor\}, \{Hersonissos, Bungalows, SwimmingPool\},$
$\{Hersonissos, Bungalows, Outdoor\}, \{Ammoudara, Bungalows, Jacuzzi\}.$

Rather than being explicitly enumerated, the 96 valid compound terms can be algebraically specified. In this way, the specification of the desired compound terms can be done in a systematic, gradual, and easy manner. For example, the following plus-product operation can be used: $\oplus_P(Location, Accommodation, Facilities)$ where

$$\begin{aligned} P = \quad & \{\{Heraklion, Furn.Appartments\}, \{Heraklion, Rooms\}, \\ & \{Ammoudara, Furn.Appartments\}, \{Ammoudara, Rooms\}, \\ & \{Hersonissos, Furn.Appartments\}, \{Ammoudara, Bungalows, Jacuzzi\}, \\ & \{Hersonissos, Rooms, Indoor\}, \{Hersonissos, Bungalows, Outdoor\}\} \end{aligned}$$

Note that the compound terms in P are only 8. Alternatively, the same result can be obtained more efficiently through the expression: $(Location \ominus_N Accommodation) \oplus_P Facilities$, where

$$\begin{aligned} N = \quad & \{\{Heraklion, Bungalows\}\}, \text{ and} \\ P = \quad & \{\{Hersonissos, Rooms, Indoor\}, \{Hersonissos, Bungalows, Outdoor\}, \\ & \{Ammoudara, Bungalows, Jacuzzi\}\} \end{aligned}$$

Note that now the total number of compound terms in P and N is just 4. In summary, the faceted taxonomy of our example, includes 13 terms, 890 compound terms, and 96 valid compound terms which can be specified by providing only 4 (carefully selected) compound terms and an appropriate algebraic expression.

Consider now the additional facet *Season* as illustrated in Figure 3, and suppose that $\{Bungalows, Winter\}$ is an invalid combination of compound terms between the previous compound taxonomy $(Location \ominus_N Accommodation) \oplus_P Facilities$ and the basic compound taxonomy *Season*. Then, the designer can declare the expression $((Location \ominus_N Accommodation) \oplus_P Facilities) \ominus_{N'} Season$ where $N' = \{\{Bungalows, Winter\}\}$. Note that the total number of compound terms in N, P, and N' is 5. The number of valid compound terms is 530. The same result could be obtained through the less efficient operation $\oplus_{P'}(Location, Accommodation, Facilities, Season)$ where:

$$\begin{aligned} P' = \quad & \{\{Heraklion, Furn.Appartments, Allyear\}, \{Heraklion, Rooms, Allyear\}, \\ & \{Ammoudara, Furn.Appartments, Allyear\}, \{Ammoudara, Rooms, Allyear\}, \\ & \{Hersonissos, Furn.Appartments, Allyear\}, \{Ammoudara, Bungalows, Jacuzzi, Summer\}, \\ & \{Hersonissos, Rooms, Indoor, Allyear\}, \{Hersonissos, Bungalows, Outdoor, Summer\}\} \end{aligned}$$

In this case the number of compound terms in P' is 8.

We will now give an example of an expression which includes a *minus-self-product* operation. Consider the faceted taxonomy of Figure 9. The user can declare the expression: $Location \oplus_P (\overset{*}{\ominus}_N (Sports))$ where

$$\begin{aligned} N = \quad & \{\{SeaSports, WinterSports\}\}, \text{ and} \\ P = \quad & \{\{Crete, SeaSki, Windsurfing\}, \{Olymbus, SnowSki\}\} \end{aligned}$$

Note the the expression $\overset{*}{\ominus}_N (Sports)$ results in the compound taxonomy shown in Figure 7.

6 Algorithms

In this Section, we present three algorithms. The first checks the validity of a compound term according to a (well-formed) expression e. The second checks if an expression e is well-formed, and the third computes the valid compound terms between two facets according to a (well-formed) expression e.

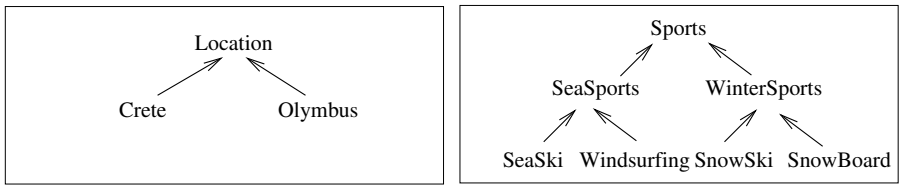

Figure 9: Another faceted taxonomy

6.1 Checking the Validity of a Compound Term

We now turn to the problem of checking whether an arbitrary compound term s ($s \in P(\mathcal{T})$) belongs to the compound terminology S_e of a given expression e. The straightforward way to achieve this, is to first compute and store the compound terminology S_e and then to check whether $s \in S_e$. However, the number of computations needed for computing S_e, as well as the storage requirements, may be very large. An alternative which we choose is to develop an algorithm which can check whether $s \in S_e$ *without* having to compute S_e. Consequently, only the expression e must be stored.

Below we present the algorithm $IsValid(e, s)$ which takes as arguments a (well-formed) expression e and a compound term s, and returns TRUE if $s \in S_e$ and FALSE otherwise (i.e. if $s \notin S_e$).

To present the algorithm we need some more notations. Let e be an expression over a facet set $\{F_1, ..., F_k\}$. The *facets* of e, denoted by $F(e)$, are defined as: $F(e) = \{F_i \mid F_i \text{ appears in } e\}$. Clearly, $F(e) \subseteq \{F_1, ..., F_k\}$. We shall denote by $F(t)$ the facet to which a term $t \in \mathcal{T}$ belongs, e.g. in Figure 1, we have $F(Crete) = Location$ and $F(SeaSki) = Sports$.

Algorithm 6.1 $IsValid(e, s)$
Input: An expression e and a compound term $s \subseteq \mathcal{T}$
Output: TRUE, if s belongs to S_e, or
FALSE, otherwise

if $s = \emptyset$ then return (TRUE)
If $\exists\, t \in s$ such that $F(t) \notin F(e)$, then return(FALSE)
if s is singleton then return(TRUE)
case(e) {
$\oplus_P(e_1, ..., e_n)$:
  if $\exists\, p \in P$ such that $p \preceq s$ then return(TRUE)
  For $i = 1, ..., n$
    if $IsValid(e_i, s)$=TRUE then return(TRUE)
  return(FALSE)
$\ominus_N(e_1, ..., e_n)$: if $\exists n \in N$ such that $s \preceq n$ then return(FALSE)
  For $i = 1, ..., n$
    Let $s_i = \{t \in s \mid F(t) \in F(e_i)\}$
    if $IsValid(e_i, s_i)$=FALSE then return(FALSE)
  return(TRUE)
$\overset{*}{\oplus}_P\ (T_i)$: if $\exists p \in P$ such that $p \preceq s$ then return(TRUE)
  if $\exists t \in \mathcal{T}_i$ s.t. $\{t\} \preceq s$ (i.e. $s \in T_i$) then return(TRUE)
  else return(FALSE)
$\overset{*}{\ominus}_N\ (T_i)$: if $\exists n \in N$ such that $s \preceq n$ then return(FALSE)
  else return(TRUE)
T_i: if $\exists t \in \mathcal{T}_i$ s.t. $\{t\} \preceq s$ (i.e. $s \in T_i$) then return(TRUE)
  else return(FALSE)
}

The algorithm is based on the parse tree of the expression e. For example, consider the faceted taxonomy of Figure 8 and assume that the desired compound taxonomy is defined by the expression $e = (Location \ominus_N Accommodation) \oplus_P Facilities$, where

$$N = \quad \{\{Heraklion, Bungalows\}\}$$

$$P = \{\{Hersonissos, Rooms, Indoor\}, \{Hersonissos, Bungalows, Outdoor\}, \{Ammoudara, Bungalows, Jacuzzi\}\}$$

Figure 10 shows the parse tree of this expression.

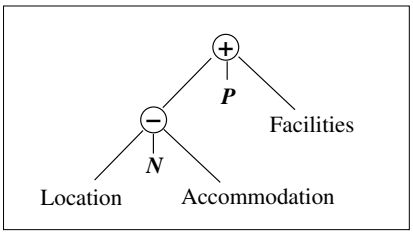

Figure 10: The parse tree of an expression

Then, it holds:
$IsValid(e, \{Hersonissos, Bungalows, SwimmingPool\})$=TRUE
$IsValid(e, \{Heraklion, Furn.Appartments\})$=TRUE
$IsValid(e, \{Hersonnissos, Bungalows, Jacuzzi\})$=FALSE

The trace of the execution of the call $IsValid(e, \{Hersonissos, Bungalows, SwimmingPool\})$ is:

```
call IsValid(e, {Hersonissos, Bungalows, SwimmingPool})
/* ∃p ∈ P s.t. p ⪯ {Hersonissos, Bungalows, SwimmingPool} */
   return(TRUE)
```

The trace of the execution of the call $IsValid(e, \{Heraklion, Furn.Appartments\})$ is:

```
call IsValid((Location ⊖N Accommodation) ⊕P Facilities, {Heraklion, Furn.Appartments})
/* ∄p ∈ P s.t. p ⪯ {Heraklion, Furn.Appartments} */
   call IsValid((Location ⊖N Accommodation), {Heraklion, Furn.Appartments})
      /* ∄n ∈ N s.t. {Heraklion, Furn.Appartments} ⪯ n */
      call IsValid(Location, {Heraklion})
         return(TRUE)
      call IsValid(Accommodation, {Furn.Appartments})
         return(TRUE)
      return(TRUE)
   return(TRUE)
```

Additionally, we give the trace of the execution of the call $IsValid(e, \{Hersonnissos, Bungalows, Jacuzzi\})$

```
call IsValid((Location ⊖N Accommodation) ⊕P Facilities, {Hersonnissos, Bungalows, Jacuzzi})
/* ∄p ∈ P s.t. p ⪯ {Hersonissos, Bungalows, Jacuzzi} */
    call IsValid((Location ⊖N Accommodation), {Hersonnissos, Bungalows, Jacuzzi})
        /* F(Jacuzzi) = Facilities ∉ F(Location ⊖N Accommodation) = {Location, Accomodation} */
        return(FALSE)
    call IsValid(Facilities, {Hersonnissos, Bungalows, Jacuzzi})
        /* F(Hersonissos) = Location ∉ F(Facilities) = {Facilities} */
        return(FALSE)
    return(FALSE)
```

Let e be an expression and s a compound term. The following proposition expresses that if $IsValid(e, s)$ =TRUE then s belongs to the compound terminology of e, or in other words that the Algorithm $IsValid(e, s)$ is correct.

Prop. 6.1 $IsValid(e, s)$ =TRUE iff $s \in S_e$

Let e be an expression, let $\mathcal{P}$ be the union of all P parameters appearing in e, and let $\mathcal{N}$ be the union of all N parameters appearing in e. The following proposition proposition indicates that the complexity of $IsValid(e, s)$ is polynomial with respect to $|\mathcal{T}|$, $|s|$, and $|\mathcal{P} \cup \mathcal{N}|$.

Prop. 6.2 Let e be an expression and let $s \subseteq \mathcal{T}$. The complexity of $IsValid(e, s)$ is $O(|\mathcal{T}|^3 * |s| * |\mathcal{P} \cup \mathcal{N}|)$.

6.2 Checking Well-Formedness of an Expression

Up to now, we have assumed that the expression e of compound taxonomies defined by the designer is well-formed. The following algorithm checks if the expression e is indeed well-formed. Specifically, $wellFormed(e)$ returns TRUE if e is well-formed. Otherwise, it returns FALSE. The algorithm $wellFormed(e)$ calls the algorithm $IsValid(e, s)$ presented in subsection 6.1. To simplify the algorithm, we assume that we have already checked that each basic compound terminology T_i, $i = 1, ..., k$, appears in e at most once.

Algorithm 6.2 $wellFormed(e)$
Input: An expression e
Output: TRUE, if e is well-formed, or FALSE, otherwise

case(e) {
$\oplus_X(e_1, ..., e_n)$ or $\ominus_X(e_1, ..., e_n)$:
  For $i = 1, ..., n$
    if $wellFormed(e_i)$=FALSE then return(FALSE)
    For all $s \in X$
      if $\exists t \in s$ s.t. $F(t) \notin F(e)$ then return(FALSE) /* $s \not\subseteq \mathcal{T}_e$ */
      Let $s_i = \{t \in s \mid F(t) \in F(e_i)\}$
      if $s_i = s$ then return(FALSE) /* $s \subseteq \mathcal{T}_{e_i}$ */
      if $IsValid(e_i, s_i)$=FALSE then return(FALSE)
  return(TRUE)
$\overset{*}{\oplus}_X(T_i)$ or $\overset{*}{\ominus}_X(T_i)$:
  For all $s \in X$
    if $\exists t \in s$ s.t. $t \notin \mathcal{T}_i$ (i.e. $s \notin \overset{*}{\oplus}(T_i)$) then return(FALSE)
    if $\exists t \in \mathcal{T}_i$ s.t. $\{t\} \preceq s$ (i.e. $s \in T_i$) then return(FALSE)
  return(TRUE) /* $\forall s \in X$, $s \in \overset{*}{\oplus}(T_i) - T_i$ */
T_i: return(TRUE)
}

For example, consider the faceted taxonomy of Figure 8 and consider the expression $e = (Location \ominus_N Accommodation) \oplus_P Facilities$, where $N = \{\{Heraklion\}\}$ and $P = \{\{Hersonissos, Rooms, Indoor\}\}$. It holds $wellFormed(e)$=FALSE. This is because $N \not\subseteq G_{Location,Accommodation}$. In particular, the trace of the execution of the call $wellFormed(e)$ is the following:

```
call wellFormed(e)
    call wellFormed(Location ⊖_N Accommodation)
        call wellFormed(Location)
            return(TRUE)
        /* s_1 = s = Heraklion */
        return(FALSE)
    return(FALSE)
```

Consider now the expression $e = Location \ominus_N Accommodation$, where $N = \{\{Heraklion, Bungalows\}\}$. It holds $wellFormed(e)$=TRUE. The trace of the execution of the call $wellFormed(e)$ follows:

```
call wellFormed(e)
    call wellFormed(Location)
```

```
        return(TRUE)
    call IsValid(Location, {Heraklion})
        return(TRUE)
    call wellFormed(Accommodation)
        return(TRUE)
    call IsValid(Accommodation, {Bungalows})
        return(TRUE)
    return(TRUE)
```

Consider now the expression $e = (Location \ominus_N Accommodation) \oplus_P Facilities$, where $N = \{\{Heraklion, Bungalows\}\}$ and $P = \{\{Heraklion, Bungalows, Indoor\}\}$. Here we have $wellFormed(e) =$ FALSE because $P \not\subseteq G_{S_{e_1}, Facilities}$, where $e_1 = Location \ominus_N Accommodation$. The trace of the execution follows:

```
call wellFormed(e)
  call wellFormed(Location ⊖_N Accommodation)
    call wellFormed(Location)
      return(TRUE)
    call IsValid(Location, {Heraklion})
      return(TRUE)
    call wellFormed(Accommodation)
      return(TRUE)
    call IsValid(Accommodation, {Bungalows})
      return(TRUE)
    return(TRUE)
  call IsValid(Location ⊖_N Accommodation, {Heraklion, Bungalows})
    return(FALSE)
  return(FALSE)
```

6.3 Computing the Valid Compound Terms between two Facets

Let an expression e of compound taxonomies, and let $F_i = (\mathcal{T}_i, \leq)$, $F_j = (\mathcal{T}_j, \leq)$ be two different facets with corresponding basic compound taxonomies T_i, T_j, respectively. We would like to compute the valid compound terms in $T_i \oplus T_j$. Specifically, let S_e be the compound terminology of e. We would like to compute $S_e \cap T_i \oplus T_j$. This is achieved through the algorithm $valid\_set(e, F_i, F_j)$, presented below.

Algorithm 6.3 $valid\_set(e, F_i, F_j)$
Input: An expression e and facets F_i, F_j s.t. $F_i \neq F_j$
Output: The valid compound terms in $T_i \oplus T_j$ (i.e., $S_e \cap T_i \oplus T_j$)

```
if F_i ∉ F(e) or F_j ∉ F(e) return({})
case(e) {
⊕_P(e_1, ..., e_n):   For l = 1, ..., n
                          if F_i, F_j ∈ F(e_l) then return(valid_set(e_l, F_i, F_j))
                      Let P' = {s_i ∪ s_j | ∃s ∈ P, ∅ ≠ s_i, s_j ⊆ s, s_i ∈ T_i, s_j ∈ T_j}
                      return(T_i ⊕_P' T_j)
⊖_N(e_1, ..., e_n):   For l = 1, ..., n
                          if F_i, F_j ∈ F(e_l) then return(valid_set(e_l, F_i, F_j))
                      Let N' = N ∩ (T_i ⊕ T_j)
                      return(T_i ⊖_N' T_j)
}
```

Obviously, it holds $valid\_set(e, F_i, F_j) \subseteq S_e$.

As an example, consider the faceted taxonomy of Figure 8. Assume that the designer declares the expression $e = (Location \ominus_N Accommodation) \oplus_P Facilities$, where

$$N = \quad \{\{Heraklion, Bungalows\}\}, \text{ and}$$

$$P = \quad \{\{Hersonissos, Rooms, Indoor\}, \{Hersonissos, Bungalows, Outdoor\}, \{Ammoudara, Bungalows, Jacuzzi\}\}$$

The algorithm $valid\_set(e, Location, Facilities)$ will return the compound terminology of $Location \oplus_{P'} Facilities$, where

$$P' = \{\{Hersonissos, Indoor\}, \{Hersonissos, Outdoor\}, \{Ammoudara, Jacuzzi\}\}$$

As another example, consider the faceted taxonomy of Figure 9. Assume that the designer declares the expression $e = Location \oplus_P (\overset{*}{\ominus}_N (Sports))$, where

$$N = \quad \{\{SeaSports, WinterSports\}\}, \text{ and}$$
$$P = \quad \{\{Heraklion, SeaSki, Windsurfing\}, \{Olymbus, SnowSki\}\}$$

The algorithm $valid\_set(e, Location, Sports)$ will return the compound terminology of $Location \oplus_{P'} Sports$, where

$$P' = \{\{Heraklion, SeaSki\}, \{Heraklion, Windsurfing\}, \{Olymbus, SnowSki\}\}$$

7 Generating Navigation Trees

Let e be an expression over a faceted taxonomy $\mathcal{F}=(\mathcal{T}, \leq)$ that defines the desired compound taxonomy $(S_e, \preceq)$. In this section we describe a method for deriving a *navigation tree* for $(S_e, \preceq)$, that can be used during the following activities:

- *Indexing* the objects of the domain. This tree can speed up the indexing process and prevent indexing errors.
- *Browsing.* This tree can aid the user to reach the objects that satisfy a given information need.
- *Testing* whether the compound taxonomy contains only the desired set of compound terms.

Intuitively, a *navigation tree* is a tree where each node n corresponds to a valid compound term s (in the sense that both n, s index the same objects). Moreover, the navigation tree contains nodes that enable the user to start browsing in one facet and then cross to another, and so on, until reaching the desired level of specificity.

Let us now introduce some notations. Given a term t, we denote by Brd(t) the set of all terms that are broader than t, i.e. Brd(t) = $\{t' \mid t \leq t'\}$. Given a compound term $s = \{t_1, ..., t_k\}$, let $Brd(s)$ be the set of all terms that are broader than some term t_i, $i = 1, ..., k$, i.e. $Brd(s)$ = Brd(t_1) ∪...∪ Brd(t_k). The navigation tree (N, R) that we construct has the following property:

> For each valid compound term s that contains at most one term from each facet, the navigation tree has a path (starting from the root) for each *topological sort*[5] of the terms of the directed acyclic graph $(Brd(s), \leq)$.

Intuitively, the last node n of each such path corresponds to the valid compound term s. For example, consider the faceted taxonomy shown in Figure 11(top), and suppose that $\{Crete, SeaSports\}$ is a valid compound term. The navigation tree in this case will include the following paths:

Location.Islands.Crete.Sports.SeaSports
Location.Islands.Sports.Crete.SeaSports
Location.Islands.Sports.SeaSports.Crete
Location.Sports.Islands.SeaSports.Crete
...
...
Sports.SeaSports.Location.Islands.Crete

[5] *Topological sort* of a set of terms is a sort that respects the partial order of the terms. That is, if $t, t' \in Brd(s)$ and $t \leq t'$, then t should always appear to the left of t' in the topological sort.

The first path above is interpreted as follows: The first term *Location* indicates that we begin browsing the facet *Location*. In this facet we focus successively on $Islands$, and then on $Crete$. After $Crete$, we decide to cross over to facet *Sports*, where we focus on $SeaSports$. All other paths are interpreted similarly. Note that the last node of all these paths corresponds to the compound term $\{Crete, SeaSports\}$.

As a further user aid whenever facet crossing occurs, a new node is created which presents the name of the facet that we are crossing to, prefixed by the string "by". This facet crossing mechanism corresponds to the use of the so-called "guide terms" for thesaurus expansion.

Each node n of the navigation tree is associated with a triple $(s(n), Fc(n), name(n))$ where:

- $s(n)$ is the *valid compound term* corresponding to n.
- $Fc(n)$ is a term, called the *focus term* of n.
  The focus term of a node n is a distinguished term among those that appear in $s(n)$ such that the children of n that are not used for facet-crossing, are the immediate narrower terms of $Fc(n)$. This means that from n, we either proceed to a different facet, or we expand $Fc(n)$ with its immediate narrower terms.
- $name(n)$ is a *name* for n.
  The name of a node is a term used for presenting the node at the user interface. It coincides with the focus term of n, unless n is a node for facet crossing. In the latter case the name of n is the name of the top term[6] of the facet we are crossing to, prefixed by the string "by".

The precise definition of the navigation tree that we construct is given below:

Def 7.1 The *navigation tree* of an expression e is a directed acyclic graph (N, R), where N is the set of *nodes* and R is the set of *edges*. Each node $n \in N$ is associated with (i) a valid compound term $s(n)$, (ii) a term $Fc(n) \in s(n)$, called the *focus term* of n, and (iii) a string $name(n)$, called the *name* of n.

There is a set of initial nodes n_i, for $i = 1, ..., k$, where $s(n_i) = \{top(F_i)\}$, $Fc(n_i) = top(F_i)$, and $name(n_i) = top(F_i)$.

For each edge in R from a node n to a node n', one of the following cases hold:

1. $s(n') = (s(n) - Fc(n)) \cup \{t\}$, $Fc(n') = t$, $name(n') = t$, and t is an immediate narrower term of $Fc(n)$.
2. $s(n') = s(n) \cup \{top(F_i)\}$, $Fc(n') = top(F_i)$, $name(n') =$"by"$+top(F_i)$, and $s(n) \cap \mathcal{T}_i = \emptyset$.
3. $s(n') = s(n)$, $Fc(n') \neq Fc(n)$, and $name(n') =$"by"$+top(F_i)$, where $F_i = F(Fc(n'))$[7].

The navigation tree starts from a set of initial nodes corresponding to the top term of each facet. Specifically, for each facet F_i, we create an initial node whose corresponding compound term is $\{top(F_i)\}$; we set as name and focus term of each such node the term $top(F_i)$. Case 1. of Definition 7.1 corresponds to the expansion of the focus term by its narrower terms. Therefore, $Fc(n')$ and $name(n')$ are set to an immediate narrower term than $Fc(n)$. Case 2. corresponds to facet crossing with a facet that has not yet been crossed to in the path from an initial node to n. $Fc(n')$ is set to the top term of that facet and $name(n')$ is set to the top term of that facet, prefixed by "by". Case 3. corresponds to facet crossing with a facet that has already been crossed. Therefore, $Fc(n')$ is set to the term in $s(n)$ that belongs to that facet. $name(n')$ is set to the top term of that facet, prefixed by "by".

There are two approaches to deriving the navigation tree. The first approach is to generate a "complete" static navigation tree through an algorithm which takes as input the expression e and

[6] Without loss of generality, we assume that each facet F_i has a greatest term with respect to subsumption which we denote by $top(F_i)$.

[7] Recall that by $F(t)$, we denote the facet that a term $t \in \mathcal{T}$ belongs to.

returns a navigation tree. The second approach is to design a mechanism that generates the navigation tree dynamically during browsing.

In Algorithm $NavTreeInit()$, we present the initialization step of the navigation tree, i.e. the creation of an initial node for each facet. In Algorithm $CreateChildren(n)$, we present the steps for creating the children of a node n[8]. These steps can be synthesized (in a depth-first-search manner) to get an algorithm that constructs the entire navigation tree. The algorithms use the function `IsValid`(e, s) which returns `True`, if s is a valid compound term according to e and `False`, otherwise. The procedure `createNode` $(name(n), s(n), Fc(n))$ creates a node with the given parameters. The function $Nar(t)$ returns the immediately narrower terms of t. The procedure `AddChild` (n, n') makes n' child of n. An algorithm that constructs the entire navigation tree in a depth-first-search manner is given in Appendix B.

Algorithm 7.1 $NavTreeInit()$
Output: An initial node for each facet

```
// Initialization: Creation of an initial node for each facet
For each i = 1, ...k
   n_i = createNode( top(F_i), {top(F_i)}, top(F_i))
return({n_1, ..., n_k})
```

Algorithm 7.2 $CreateChildren(n)$
Input: A node n of the navigation tree
Output: The children of n

```
      Let children := {}
B.1   // Creating the children of a node on the basis of the focus term
      For each t ∈ Nar(Fc(n))
         Let s' := (s(n) − Fc(n)) ∪ {t}
         If IsValid(e, s') then
            n' := createNode(t, s', t)                          // Case 1.
            AddChild(n, n')
            Let children := children ∪ {n'}

B.2   // Creating the children of a node for "facet crossing"
      For each F_i ∈ F(e) − F(Fc(n))
         Let s_i := s(n) ∩ T_i
         If s_i = ∅ then
            Let s' := s(n) ∪ {top(F_i)}
            If IsValid(e, s') then
               n' := createNode("by" + top(F_i), s', top(F_i) )   // Case 2.
               AddChild(n, n')
               Let children := children ∪ {n'}
         else
            If s_i = {t_i} and ∃t' ∈ Nar(t_i): IsValid( e, (s(n) − {t_i}) ∪ {t'}) then
               n' := createNode("by"+top(F_i), s(n), t_i)       // Case 3.
               AddChild(n, n')
               Let children := children ∪ {n'}
      return(children)
```

Figure 11 shows a part of the navigation tree that is generated by this algorithm for the faceted taxonomy shown at the top of the Figure and expression $e = Sports \ominus_N Location$, where $N = \{\{WinterSports, Islands\}, \{SeaSports, Olympus\}\}$. In the navigation tree, each node n is presented by its name, $name(n)$. For example, the node n_{22} has $name(n_{22}) =$ `Mainland`, $s(n_{22}) =$ {`Sports`, `Mainland`}, $Fc(n_{22}) =$ `Mainland`. The nodes n_{23} and n_{27} are generated by part B.1 of the algorithm $CreateChildren(n)$, while node n_{30} is generated by part B.2.

[8]With this algorithm, we do not attempt to propose a user interface, but just to give the primitives for designing one.

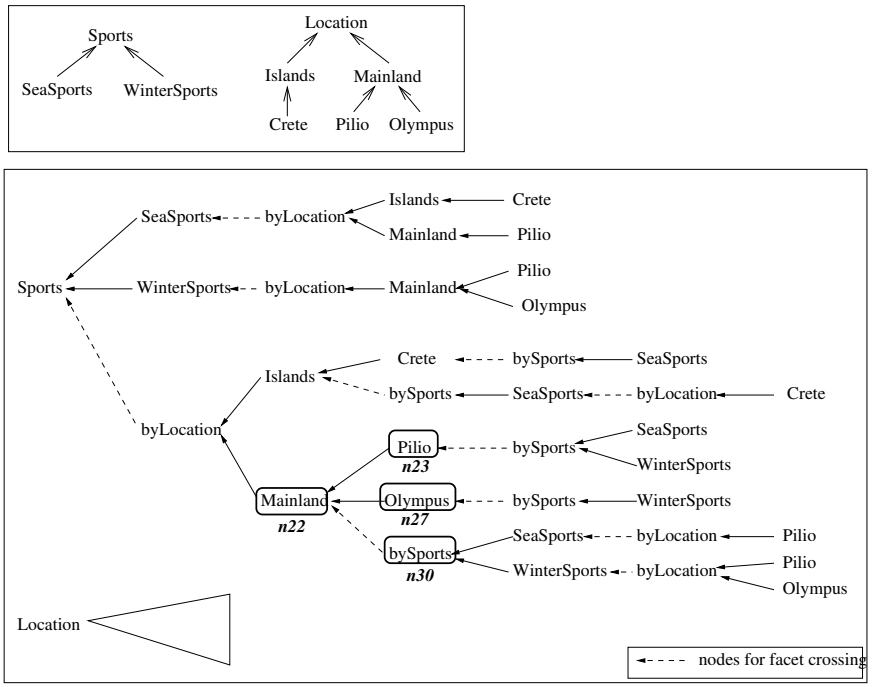

Figure 11: Example of a navigation tree

Note that the navigation tree of Figure 11 is "complete" for browsing, as it supports complete facet crossing. In a single taxonomy, one browses until reaching the desired terms. However, single taxonomies may not be "complete" for browsing, in the sense that it may not be possible from any node to switch to a different aspect of interest (facet). This implies that if an aspect of interest is deeper in the hierarchy, the user may not know which path to follow to reach the desired node in the taxonomy. Thus the user may follow many "irrelevant" paths which involves a lot of backtracking and alternative route selection. On the other hand, generated navigation trees in faceted taxonomies can be "complete" with little cost, as they are created dynamically upon user request. Thus users can reach the desired node, without need for backtracking. Note that "complete" pre-determined single taxonomies are not practical, as the number of nodes grows exponentially with the number of aspects.

For example, consider the single taxonomy of Figure 12, and assume that the user wants to find all hotels that offer sea-sports facilities. Then, the user has to follow all paths: $Hotels > Islands > Crete > Seasports$, $Hotels > Mainland > Pilio > SeaSports$, and $Hotels > Mainland > Olympus$ in order to find that sea-sports are offered only by hotels in Crete and Pilio. Note that the last path is "irrelevant". In contrast, in the derived navigation tree of Figure 11, the user just follows the path $Sports > SeaSports$. All information under this node is relevant to his/her query.

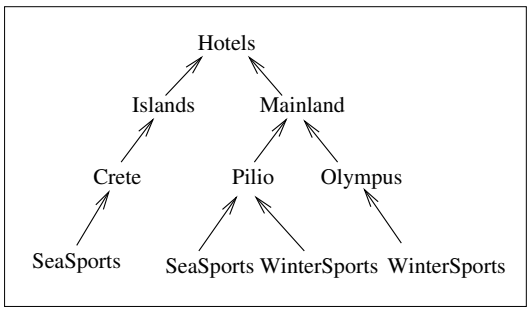

Figure 12: Example of an "incomplete" single taxonomy

8 The Design Process

Figure 13 illustrates the process for designing a compound taxonomy. The first task is to select the facets. The second is to define a taxonomy for each facet, thus defining the faceted taxonomy $\mathcal{F}$. The third task is to formulate an expression e over $\mathcal{F}$ that specifies the desired compound taxonomy. The fourth task is to test the compound taxonomy defined by e, and the final (and incessant) task is to maintain the compound taxonomy. Below we discuss each step in more detail.

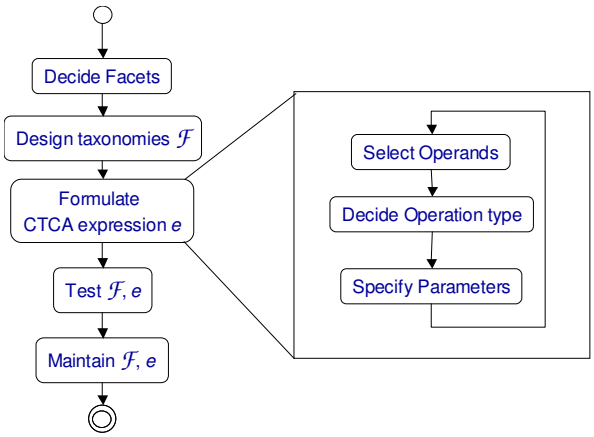

Figure 13: The process for designing a compound taxonomy

8.1 Deciding Facets

The faceted classification paradigm suggests to view the knowledge domain through a set of different planes of understanding, else called aspects, points-of-view, or *facets*. The division of the knowledge domain according to facets resembles up to some degree the division of our physical space into dimensions. For instance, we could construe a scheme comprising 3 facets A, B and C, as an orthocanonical system with three axes A, B and C. Now the index of an object o with respect to this faceted taxonomy corresponds to the coordinates of a point o in the 3D space (see Figure 14). Note that from a mathematical point of view, a faceted space is more general than the physical space, not only because the physical space has only three dimensions, but also because each axe of the physical space is a linear order of values, while each axe of a faceted space can be a partial order of values (i.e. a taxonomy).

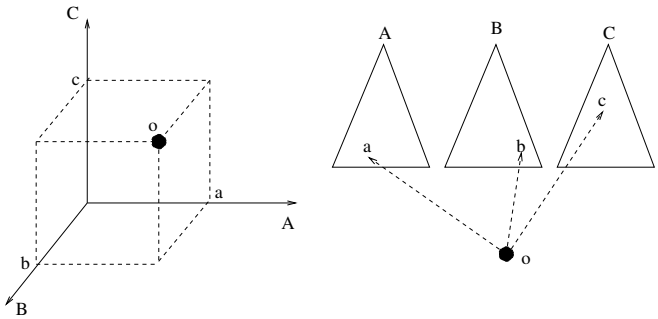

Figure 14: Physical versus faceted space

It has been written several times that each facet should describe the domain from a different, preferably *orthogonal*, point of view. We could say that two facets A and B are orthogonal if the probability that a random term a of A applies on a random object o of the domain is independent of the probability that a random term b of B applies on o. Of course, orthogonality is by no means a strict rule. It is rather a suggestion because in practice, several times we may have to employ facets and terms which are not actually orthogonal, but they are still good choices due to other kind of reasons, e.g. they may

be very familiar or widely used within a community of users (or they may correspond to standards, etc).

Even if we have a set of "orthogonal" facets, in the broad sense, there may be terms which are not actually independent. For instance, suppose the facets *Location*, *Sports*, and *Facilities*. Notice that the term *SmokeSauna* (from *Facilities*) actually implies the term *Finland* (from *Location*). Hence we could write $P(Finland|SmokeSauna) = 1$. On the other hand, the term *SeaSports* actually excludes all terms (from *Location*) that denote mountainous regions, such as *Olympus*. For example we can write $P(Olympus|SeaSports) = 0$.

At last we should mention that there is not any specific recipe for choosing the best facets to be defined for an application domain. After all, there is not even a recipe for defining the entity types of an Entity-Relationship diagram [7]. Section 10 further discusses this issue.

8.2 Designing Taxonomies

For constructing a taxonomy, a useful/helpful hint is to consider that each term actually specifies a set of objects of the domain and that each subsumption relationship $a \leq b$ actually specifies an inclusion relationship between the objects described by a and those described by b (e.g. see (a) and (b) of Figure 15). If apart from inclusion relationships we would like to state that the intersection of the extensions of two (or more) particular terms is not empty (in the knowledge domain), then we could either introduce a new term that is subsumed by these two terms, or employ a self-product operation. For example, a self-product operation allows modeling the case depicted in Figure 15.(c) without having to introduce a new term in the taxonomy.

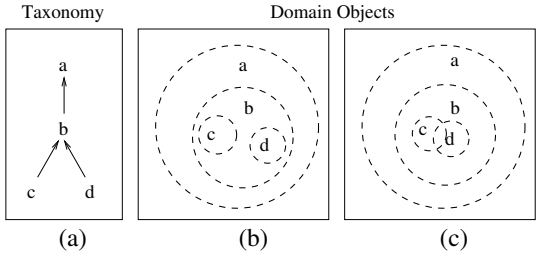

Figure 15: Taxonomies and domain objects

8.3 Formulating the Expression

So, let's suppose that V denotes the subset of $\mathcal{P}(\mathcal{T})$ that contains the valid (meaningful) compound terms in the knowledge domain. As we have shown in Section 3, V is an *o-filter*. A designer can use CTCA in order to specify the set V in a flexible and gradual manner, without having to provide explicitly every element of V (the manual specification of the elements of V would be a formidably laborious task). Specifically, what he/she has to do is to formulate a CTCA expression e such that $S_e = V$. Note that if we had a materialization of $\mathcal{F}$ that was "complete" - in the sense that the extension of every valid compound term is non-empty - then we could *mine* the expression e using the approach presented in [36] (this is called *expression mining*). However, we rarely have such a complete knowledge and certainly we do not have any materialization when we design the faceted taxonomy.

Note that there are several expressions e such that $S_e = V$, where V is a subset of $\mathcal{P}(\mathcal{T})$ such that $V = Br(V)$. Specifically, in [36], we show that for k facets, there are more than $k!$ different expression forms (i.e. parse trees). Additionally, we show that for any faceted taxonomy $\mathcal{F}= (\mathcal{T}, \leq)$ and set $V \subseteq P(\mathcal{T})$ that is an *o-filter*, there is an expression e of *any* expression form over $\mathcal{F}$ s.t. $S_e = V$. Let $E(V)$ denote the set of all expressions over $\mathcal{F}$ such that $S_e = V$. So, the goal of the designer is to formulate one expression of $E(V)$. The above discussion raises the following questions: (a) what is the more convenient (and systematic) method with which a designer can reach an expression in $E(V)$, and (b) which expressions of $E(V)$ are more "scalable", i.e. can follow the evolution of $\mathcal{F}$ (specifically,

the updates of the taxonomies, as well as the addition of new facets)? These issues are subject of ongoing research. However, below we provide some tips that originate from our experience so far.

8.3.1 Selecting the Operation Type and Specifying the Parameters

Consider the compound taxonomies $S_1, ..., S_n$ and suppose that we have to specify those elements of $G_{S_1,...,S_n}$ that are valid. If the majority of the elements of $G_{S_1,...,S_n}$ are valid then it is better for the designer to use a *minus-product* operation so as to specify only the invalid compound terms because they are less in number than the set of valid compound terms, so he/she will have to provide a less number of parameters. Concerning the methodology for defining the set N of a minus-product operation, it is more efficient for the designer to put in N "short" compound terms that consist of "broad" (w.r.t. $\leq$) terms. The reason is that from such compound terms a large number of new invalid compound terms can be inferred. In particular, what we are looking for is a Sperner system[9] of the maximal invalid compound terms.

Conversely, if the majority of the genuine compound terms are invalid, then it is better for the designer to employ a *plus-product* operation, so as to specify only the valid compound terms (with a comparatively smaller parameter set). Concerning the methodology for defining the set P of a plus-product operation, it is more efficient for the designer to put in P "long" compound terms that consist of "narrow" terms, since from such compound terms a large number of new valid compound terms can be inferred. In particular, what we are looking for is a Sperner system of the minimal valid compound terms.

Minus-product operations "follow better" the evolution of taxonomies, if they evolve in a top-down manner (addition of leaves). On the other hand, if taxonomies evolve in a bottom-up manner (e.g. assume the case where we have a rather fixed set of leaves and new terms are added on top of these), then plus-product operations follow better evolution.

8.4 Testing the expression

The designer can test the compound taxonomy $(S_e, \preceq)$ by browsing the derived navigation tree defined in Section 7. Moreover, the designer can use the algorithm $valid(e, s)$ to check if a desired compound term s is valid, or the algorithm $valid\_set(e, F_i, F_j)$ in order to test the valid compound terms in $T_i \oplus T_j$. If the designer realizes that the compound taxonomy does not contain the desired set of compound terms, then he can update the expression e and then test the outcome again, and so on.

8.5 Supporting Evolution

CTCA allows the specification of the valid and invalid compound terms of a faceted taxonomy according to the current state of affairs, in an efficient and flexible manner. Obviously, if the state of affairs changes, currently valid terms may become invalid and vice-versa. Moreover, it is possible that the faceted taxonomy is also modified, making an algebraic expression no longer well-formed.

The problem of updating automatically an algebraic expression over a faceted taxonomy, when the faceted taxonomy is modified, is treated in [35, 33]. Another related work is the one described in [36] which deals with the problem of computing the shortest (with the least storage space requirements) well-formed expression e such that $S_e = V$, where V is an *o-filter*. The algorithms provided in that paper can be exploited for supporting evolution. For instance, assume that it is desirable to add a set of compound terms S to the set of valid compound terms described by an algebraic expression e. Then, the algorithms described in [36] can be applied such that a new well-formed expression e' is computed with $S_{e'} = S_e \cup Br(S)$. Now, assume that it is desirable to remove a set of compound terms S from the set of valid compound terms described by an algebraic expression e. Then, the algorithms described in that paper can be applied such that a new well-formed expression e' is computed with $S_{e'} = S_e - Nr(S)$. These algorithms take as input a set of valid compound terms. This means that for

[9] A *Sperner system* [29] is a set system $\mathcal{N}$ such that if $X, Y \in \mathcal{N}$ and $X \neq Y$ then $X \not\subset Y$ and $Y \not\subset X$.

applying these algorithms to our case, we have to optimize them so that to avoid having to compute S_e before applying them. This is an issue for further research.

9 Implementation and Applications

A system based on CTCA, called FASTAXON [39], has already been implemented by VTT and Helsinki University of Technology (HUT). FASTAXON has been implemented as a client/server Web-based system, written in Java. The server is based on the Apache Web server, the Tomcat application server, and uses MySQL for persistent storage. The user interface is based on DHTML (dynamic HTML), JSP (Java Server Pages), and Java Servlet technologies (J2EE). The client only needs a Web browser that supports JavaScripts (e.g. Microsoft Internet Explorer 6). Figure 16 illustrates the general architecture of the system. Roughly, this system allows storing faceted taxonomies, formulating algebraic expressions over them, reasoning about the validity of compound terms, and generating navigational trees.

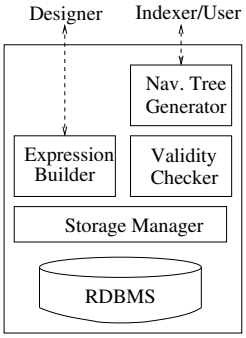

Figure 16: The architecture

The faceted taxonomy $(\mathcal{T}, \leq)$ can be stored using three tables:

```
FACETS(id:Int, name:String)
TERMS(id:Int, name:String, facetId:Int)
SUBS(termId:Int, broaderTermId:Int)
```

The first stores the names of the facets, the second stores the terms of $\mathcal{T}$, and the third the transitive reduction (Hasse diagram) of the subsumption relation $\leq$. Clearly, we also have the corresponding foreign key constraints `TERMS.facetId` $\subseteq$ `FACETS.id`, `SUBS.termId` $\subseteq$ `TERMS.id`, and `SUBS.broaderTermId` $\subseteq$ `TERMS.id`. Concerning the storage of the parameters we can distinguish two cases. If all elements of the parameters P and N of e contain at most one term from each terminology $\mathcal{T}_i$ then we can store all P and N parameters in one table `PARAMS` that has one attribute A_i for each facet F_i, $i = 1, .., k$, and $dom(A_i) = \mathcal{T}_i$. Specifically, each element $s = \{t_1, ..., t_m\}$ in P (or N) is stored as a tuple r such that[10] $r(A_{J(t_i)}) = t_i$, for each $i = 1, ..., m$. In order to speed up the lookup operations on the relation `PARAMS` (i.e. the selection queries), each operation in the parse tree of the expression e is assigned a unique identifier `opId`. This identifier is associated with each element of the parameter, P or N, of the operation. This has been implemented by adding one additional column `opId` to the relation `PARAMS`. In this way, the elements of each parameter P or N can be collected and searched more efficiently. However, in the general case where there are elements in P or N that contain more than one term from one facet (e.g. when there are one or more self-product operations), we cannot use the table `PARAMS` as defined earlier, because we would need set-valued attributes (but then the relation would not be in 1NF). For this reason we use a scheme which "simulates" set valued attributes (we assign an identifier to each value set and store the values of the set in a separate table). Specifically, we use two tables:

[10] Let $t \in \mathcal{T}_j$. We define $J(t) = j$.

PARAMS($A_1, ..., A_k$, opId)
SETS(setId:Int, termId:Int)

Let s be an element of a parameter P or N, which is stored as a tuple r in PARAMS. If there is $i = 1, ..., k$ such that $s \bigcap \mathcal{T}_i = \{t_1, ..., t_n\}$ with $n > 1$, then we create a new identifier setId for $\{t_1, ..., t_n\}$ and store setId in the appropriate attribute of r, i.e. $r(A_i) = setId$. Now the correspondence between setId and the terms $\{t_1, ..., t_n\}$ is stored in SETS.

For example, consider the expression $e = (Sports \oplus_{P1} Places) \ominus_N (\overset{*}{\oplus}_{P2} (Facilities))$ where

$$\begin{aligned} P1 &= \{\{SeaSports, Crete\}, \{WinterSports, Greece\} \\ P2 &= \{\{Indoor, Outdoor\}, \{Outdoor, Jacuzzi\}\} \\ N &= \{SeaSports, Crete, Indoor\} \end{aligned}$$

The contents of the tables PARAMS and SETS follow:

PARAMS

Sports	Places	Facilities	opId
SeaSports	Crete		o1
WinterSports	Greece		o1
		s1	o2
		s2	o2
SeaSports	Crete	Indoor	o3

SETS

setId	termId
s1	Indoor
s1	Outdoor
s2	Outdoor
s2	Jacuzzi

The system is currently under experimental evaluation. The productivity obtained using FASTAXON is quite impressive. So far, the experimental evaluation has shown that in many cases a designer can define from scratch a compound taxonomy of around 1000 indexing terms in some minutes.

Here, we would like to note that in [4], it is shown how CTCA can be efficiently represented in logic programming, providing an alternative kind of implementation.

We now describe a few applications of the algebra.

- Building and Re-engineering the Taxonomies for Catalogs

  The algebra can be used in any application that indexes objects using a controlled structured vocabulary, i.e. a taxonomy. For example, it can be used for designing compound taxonomies for products, for fields of knowledge (e.g. for indexing the books of a library), etc. Furthermore, it can be exploited for *reorganizing* the big fat single-hierarchical taxonomies found on the Web (like Yahoo! or ODP) and in Libraries (like the majority of the thesauri [17]), so as to give them a clear, compact, and scalable faceted structure but *without missing* the knowledge that is hardwired in their structure and in their pre-coordinated vocabularies. Such a reorganization would certainly facilitate their management, extension, and reuse. Furthermore, it would allow the dynamic derivation of "complete" and meaningful navigational trees for this kind of sources, which unlike the existing navigational trees of single-hierarchical taxonomies, do not present the problem of missing terms or missing relationships (for more about this problem see [8]). For example, for reusing the taxonomy of the Google directory, we now have to copy its entire taxonomy which currently consists of more than 450.000 terms and whose RDF representation[11] is a compressed file of 46 MBytes! According to our approach, we only have to *break* their pre-coordinated terms and then *partition* the resulting set of elemental terms to a set of facets, using languages like the one proposed in [30] (we shall not elaborate on this issue in this paper), and then use the algorithms for expression mining [36]. Apart from smaller storage space requirements, the resulted faceted taxonomy can be modified/customized in a more flexible and

[11] Available at http://rdf.dmoz.org/

efficient manner. Furthermore, a semantically clear, faceted structure can aid the manual or automatic construction of the inter-taxonomy mappings ([40]), which are needed in order to build mediators or peer-to-peer systems over this kind of sources [42, 41]. Figure 17 illustrates graphically this scenario.

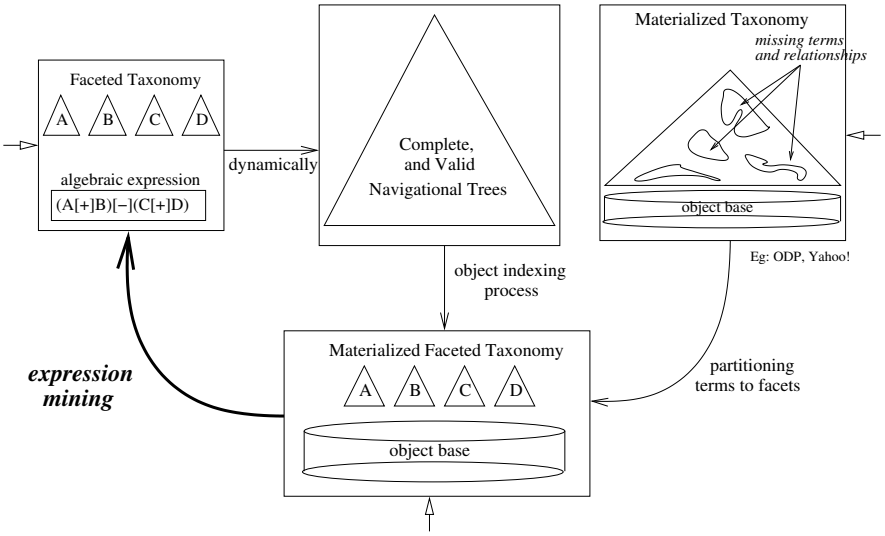

Figure 17: Some application scenarios of CTCA

- Compact Exchange

  A CTCA expression can also be used for exchanging compactly the compound terms that are extensionally valid according to a materialized faceted taxonomy (using the expression mining algorithms presented in [36]). Moreover, as showed in [32, 34], CTCA can be used for compressing a Symbolic Data Table [11].

- Configuration Management

  Our algebra can also be used for *configuration management*. Consider a product (e.g. a software module) whose functionality, or configuration, is determined by a number of parameters, where each parameter is associated with a finite number of values. However, some configurations may either make no sense, or they are not supported, or they may be "dangerous". For this purpose, the designer of the product can use an expression in order to specify all valid configurations. In this way, the user of the product can select one configuration from the valid ones. The advantage of using our approach is that the space needed for storing the valid configurations is low. This is quite important, as all information about configuration management has to be stored in the product itself, e.g. in the software module. As another example, consider a computer sales company that allows customers to order a computer with parts of their choice, through a Web portal. However, not all configurations of computer parts are valid. For example, a selected motherboard may not support all available processors, or memory types. In this case, an algebraic expression may be defined that specifies all valid configurations allowing the user to select only among these. As it is shown in [37], defining these kinds of configurations in CTCA is much economical in terms of space and time complexity than defining these in Description Logics.

- Optimizing Query Answering

  Suppose a materialized faceted taxonomy that stores a very large number of objects and users that pose conjunctive queries (i.e. compound terms) in order to find the objects of interest. The availability of an expression e that specifies the compound terms with non-empty extension would allow realizing efficiently (in polynomial time with respect to $|\mathcal{T}|$) whether a query has an empty or a non-empty answer. In this way, we can avoid performing wasteful computations for the queries whose answer will be eventually empty.

More applications are discussed in [36, 37].

10 Related Work

Although faceted classification was suggested quite long ago (by Ranganatham in the 1920s), the associated issues have not received adequate attention by the computer science community. However, there are several works about facet analysis (e.g. see [12, 44, 19, 10]). Facets have been studied in library and information science (for a review see [20]). A faceted structure for organizing an institutional website was proposed in [25]. The use of faceted taxonomies is also common among e-commerce sites [3]. Navigational faceted access structures are also proposed in [18, 46]. Below we discuss in brief the classifications schemes in library and information science and facet analysis.

In Library and Information Science we can identify three approaches to classification: enumerative, hierarchical, and analytico-synthetic (or faceted). *Enumerative classification* attempts to assign headings for every subject and alphabetically enumerates them. *Hierarchical classification* establishes logical rules for dividing topics into classes, narrower classes, and so on. *Analytico-synthetic* (or *faceted*) classification assigns terms to individual concepts and provides rules for the local cataloguer to use in constructing headings for composite subjects. Ranganathan was the first to introduce the word "facet" into library and information science in 1933. Ranganathan demonstrated that *analysis*, which is the process of breaking down subjects into their elemental concepts, and *synthesis*, the process of recombining those concepts into subject strings, could be applied to all subjects, and demonstrated that this process could be systematized. The phrase "analytico-synthetic classification" derives from these two processes: analysis and synthesis. For a wider review on the faceted access to information see [20].

As commented in [12], the basic principle of facet analysis is that concepts can be grouped using a division which is not necessarily hierarchical. In other words, subjects which have previously been subdivided by progressive hierarchical arrangement, forming the familiar "tree structures", can be looked on as patterns of horizontal division, as well as. In any area of complex ideas, by using horizontal groupings, new subjects are formed by combining sub-subjects from the various horizontal groups.
B.C. Vickery [44] compares the semantic model of human memory structures used by Linsday and Norman [19] with the analysis of subjects by facet, used by himself and others in subject classification. Lindsay and Norman described "roles which characterize parts of an event" as:

Action, Agent, Conditional, Instrument, Location,
Object, Purpose Quality, Recipient, Time

These correspond closely with some of the facets defined by Vickery [44] as being useful within a science and technology classification:

Attributes, Object, Parts, Place, Processes, Properties, Substances, Time, Recipient

Research based on facet analysis [12] has been able to define facets which may be labelled differently in different domains, but which are essentially transferable. The following examples are taken from knowledge bases built for research purposes using the hypertext system "NoteCards".

generic labelling	*catering*	*social skills*
parts	ingredients	attitudes
processes	processes	processes
procedures	recipes	procedures
agents	equipment	people
properties	characteristics	situations
products		

There are many faceted taxonomies that have been proposed for specific application domains (e.g. see Table 1). For instance, thesauri ([17]) may have facets that group the terms of the thesaurus in classes. For example, Ruben Prieto-Diaz ([23, 24]) has proposed "faceted classification" for a reusable software library. In a faceted classification scheme, the facets may be considered to be dimensions in a cartesian classification space, and the value of a facet is the position of the artifact in that dimension. For software, one might have facets with values such as "Operand", "Functionality", "Platform", "Language", etc. Prieto-Diaz claims that a fixed (and small) number of facets is sufficient for classifying all software.

Purpose:Source	*Facets*
Document Classification: Ranganathan [26]	Space, Time, Energy, Matter, Personality
Art and Architecture: AAT [16]	Associated Concepts, Physical Attributes, Styles and Periods, Agents, Activities, Materials, Objects
Science and Technology Classification:Vickery[44]	Place, Time, Attributes, Properties, Object, Parts, Processes, Substances, Recipient
Dissertation abstracts: Sugarman [31]	Operation, Object, Properties
Software classification: Prieto-Diaz [23]	Function, Object, Medium

Table 1: Examples of faceted schemes

The novelty of our approach lies in enriching a faceted scheme with an algebra for specifying the valid compound terms. This method can be used in order to construct taxonomies or thesauri, which unlike existing thesauri, do not present the problem of missing terms or missing relationships (for more about this problem, see [8]).

Coordination [14] is the act of linking together index terms to provide a more precise description of an object. Term coordination can occur at three stages: during the time of index construction, at the stage of indexing, or at retrieval time. In *pre-coordination*, term coordination occurs at the stage of index construction or indexing. In contrast, in *post-coordination*, term coordination is done by the searcher at retrieval time. Specifically, in post-coordination, indexing involves assigning terms to a document, but linking is done at search time.

A single taxonomy is by definition pre-coordinated. On the other hand, a faceted taxonomy may be applied both pre-coordinately and post-coordinately. Post-coordination has the advantage that allows one to access objects from many different access points. In contrast, pre-coordinated subject indexes have a single entry point. However, we have to note that they improve precision in those subject headings that order and relations between facets affects the meaning. In our theory, a compound term is interpreted as the conjunction of its elements, and corresponds to post-coordination. Pre-coordinated terms (e.g. $SeaSpots$) are treated in our context as simple terms.

Current interest in faceted taxonomies is also indicated by several recent projects (like FATKS[12], FACET[13], FLAMENGO[14]), and the emergence of XFML (Core-eXchangeable Faceted Metadata Language) [1]. XFML is a model to express topics organized in facets, and allows you to assign topics to any page on the web. XFML lets you publish this information in an open, XML based format. We extended XFML to XFML+CAMEL (*C*ompound term composition *A*lgebraically-*M*otivated *E*xpression *L*anguage) [2] which allows publishing and exchanging faceted taxonomies *and* expressions of the com-

[12]http://www.ucl.ac.uk/fatks/database.htm
[13]http://www.glam.ac.uk/soc/research/hypermedia/facet_proj/index.php
[14]http://bailando.sims.berkeley.edu/flamenco.html

pound term composition algebra in XML format.

The semantics of CTCA are formally defined in [37], where it has been shown that CTCA cannot be efficiently represented in Description Logics. In brief, we have shown that if we want to check the validity of compound terms using the classical reasoning services of DL [5], then we cannot represent CTCA expressions in DL in a straightforward manner. Instead, we have to either: (a) convert all minus-products and minus-self-products to equivalent plus-products and plus-self-products (and then translate the resulting plus-products and plus-self-products to DL), or (b) convert all plus-products, plus-self-products, and basic compound terminologies to equivalent minus-products and minus-self-products (and then translate the resulting minus-products and minus-self-products to DL). Recall that the reason CTCA supports both positive and negative statements (i.e. plus-(self)-products and minus-(self)-products) is to allow the designer to select at each step the most economical operation, i.e. the one that requires providing the smaller parameter. Under this assumption, it follows that the above conversion is expected to result in an expression of much larger size and thus, to a much larger in size DL theory.

11 Concluding Remarks

The novelty of our approach lies in enriching a faceted scheme with an algebra for specifying the valid compound terms. The advantages of our approach are the following:

- The algebra that we propose is quite *flexible* and quite *easy* to use. The designer does not have to write a program or to be familiar with logic-based languages. He just decides the order by which the facets appear in the expression and sets the parameters P and N which are just sets of compound terms. The simplicity of the compound terms considered (conjunctions of terms only) apart from allowing a very efficient inference mechanism, makes our approach easy to use and scalable. We believe that it can be adopted by catalog designers (librarians, etc) who are not familiar with logic-based representation languages.

- The operations are defined in a way that ensures monotonicity. This means that when the designer adds a new facet to the expression and defines the parameters P or N, he does not have to worry about invalidation of previous results.

- The compound taxonomies defined by our algebra have *low storage space* requirements. There is no need to store the compound taxonomy of an expression. Only the expression has to be stored, as we provided an *efficient* inference mechanism which can check whether a compound term belongs to the compound terminology of the expression.

- We can generate dynamically *navigation trees* which are suitable for browsing and can be also exploited during the indexing process (to aid the indexer and prevent indexing errors).

Our algebra can be used in any application that indexes objects using a controlled structured vocabulary, i.e. a taxonomy. For example, it can be used for designing compound taxonomies for products, for fields of knowledge (e.g. for indexing the books of a library), etc.

Finally, we have to note that the advantages of our approach (compactness, conceptual clarity, scalability, valid compound terms) can facilitate several other associated tasks. Specifically, they can certainly facilitate the design of *mediators* and *peer-to-peer* systems of taxonomy-based sources (using the approach presented in [42, 41], respectively), and the *personalization* of Web catalogs (using the approach presented in [30]).

References

[1] "XFML: eXchangeable Faceted Metadata Language". http://www.xfml.org.

[2] "XFML+CAMEL:Compound term composition Algebraically-Motivated Expression Language". http://www.ics.forth.gr/markup/xfml+camel.

[3] H. P. Adkisson. "Use of faceted classification", 2005. (http://www.webdesignpractices.com/navigation/facets.html).

[4] Anastasia Analyti and Ioannis Pachoulakis. "Logic Programming Representation of the Compound Term Composition Algebra". *Fundamenta Informaticae*, 2006. (accepted for publication).

[5] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *"The Description Logic Handbook: Theory, Implementation, and Applications"*. Cambridge University Press, 2003.

[6] G. H. Bordalo and B. Monjardet. "The lattice of strict completions of a finite poset". *Algebra Universalis*, 47:183–200, 2002.

[7] P. Chen. "The Entity-Relationship Model - Toward a Unified View of Data". *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[8] Peter Clark, John Thompson, Heather Holmback, and Lisbeth Duncan. "Exploiting a Thesaurus-based Semantic Net for Knowledge-based Search". In *Procs. of the 12th Conf. on Innovative Applications of AI (AAAI/IAAI'00)*, pages 988–995, 2000.

[9] W. F. Cody, J. T. Kreulen, V. Krishna, and W. S. Spangler. "The integration of business intelligence and knowledge management". *IBM Systems Journal*, 41(4):697–713, 2002.

[10] Willian Denton. "How to Make a Faceted Classification and Put It On the Web", 2003. (http://www.miskatonic.org/library/facet-web-howto.html).

[11] Edwin Diday. "An Introduction to Symbolic Data Analysis and the Sodas Software". *Journal of Symbolic Data Analysis*, 0(0), 2002. ISSN 1723-5081.

[12] Elizabeth B. Duncan. "A Faceted Approach to Hypertext". In Ray McAleese, editor, *HYPERTEXT: theory into practice, BSP*, pages 157–163, 1989.

[13] M. Eirinaki, M. Vazirgiannis, and I. Varlamis. "SEWeP: using Site Semantics and a Taxonomy to Enhance the Web Personalization Process". In *Procs. of the 9th ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining (KDD'03)*, pages 99–108, New York, NY, USA, 2003. ACM Press.

[14] A.C. Foskett. *"The Subject Approach to Information. Fourth edition"*. The Shoe String Press Inc., Hamden, Connecticut, 1982.

[15] Stephen C. Gates, Wilfried Teiken, and Keh-Shin F. Cheng. "Taxonomies by the Numbers: Building High-performance Taxonomies". In *Procs. of the 14th ACM Intern. Conf. on Information and Knowledge Management (CIKM'05)*, pages 568–577, New York, NY, USA, 2005. ACM Press.

[16] Getty Research Institute. "Art & Architecture Thesaurus". (http://www.getty.edu/research/tools/vocabulary/aat/).

[17] International Organization For Standardization. "Documentation - Guidelines for the establishment and development of monolingual thesauri", 1986. Ref. No ISO 2788-1986.

[18] F. Lima and D. Schwabe. "Application Modeling for the Semantic Web". In *Procs. of the 1st Latin American Web Congress (LA-WEB 2003)*, pages 93–102, Sanitago, Chile, 2003.

[19] P. H. Lindsay and D. A. Norman. *Human Information Processing.* Academic press, New York, 1977.

[20] Amanda Maple. "Faceted Access: A Review of the Literature", 1995. http://theme.music.indiana.edu/tech_s/mla/facacc.rev.

[21] Deborah L. McGuinness. "Ontological Issues for Knowledge-Enhanced Search". In *Proceedings of FOIS'98*, Trento, Italy, June 1998. Amsterdam, IOS Press.

[22] C.H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[23] Ruben Prieto-Diaz. "Classification of Reusable Modules". In *Software Reusability. Volume I*, chapter 4, pages 99–123. acm press, 1989.

[24] Ruben Prieto-Diaz. "Implementing Faceted Classification for Software Reuse". *Communications of the ACM*, 34(5):88–97, 1991.

[25] U. Priss and E. Jacob. "Utilizing Faceted Structures for Information Systems Design". In *Proceedings of the ASIS Annual Conf. on Knowledge: Creation, Organization, and Use (ASIS'99)*, October 1999.

[26] S. R. Ranganathan. "The Colon Classification". In Susan Artandi, editor, *Vol IV of the Rutgers Series on Systems for the Intellectual Organization of Information*. New Brunswick, NJ: Graduate School of Library Science, Rutgers University, 1965.

[27] G. Salton. *"Introduction to Modern Information Retrieval"*. McGraw-Hill, 1983.

[28] M. B. Smyth. "Power Domains". *Journal of Computer and System Sciences*, 16:23–36, 1978.

[29] Emanuel Sperner. "Ein Satz über Untermengen einer endlichen Menge". *Math. Zeitschrift*, 27:544–548, 1928.

[30] Nicolas Spyratos, Yannis Tzitzikas, and Vassilis Christophides. "On Personalizing the Catalogs of Web Portals". In *Procs. of the 15th International FLAIRS Conference (FLAIRS'02)*, pages 430–434, Pensacola, Florida, May 2002.

[31] J. H. Sugarman. *"The development of a classification system for information storage and retrieval purposes based upon a model of scientific knowledge generation"*. PhD thesis, School of Education, Boston University, 1981.

[32] Yannis Tzitzikas. "An Algebraic Method for Compressing Very Large Symbolic Data Tables". In *Procs. of the Workshop on Symbolic and Spatial Data Analysis of ECML/PKDD 2004*, Pisa, Italy, September 2004.

[33] Yannis Tzitzikas. "Updates and Revision in Faceted Taxonomies and CTCA Expressions". Technical Report 2005-11-18, TR 364, Institute of Computer Science-FORTH, November 2005.

[34] Yannis Tzitzikas. "An Algebraic Method for Compressing Symbolic Data Tables". *Journal of Intelligent Data Analysis (IDA)*, 2006. (accepted for publication).

[35] Yannis Tzitzikas. "Revising Faceted Taxonomies and CTCA Expressions". In *Procs. of the 4th Hellenic Conference on AI (SETN 2006)*, pages 600–604, Heraklion, Greece, May 2006.

[36] Yannis Tzitzikas and Anastasia Analyti. "Mining the Meaningful Term Conjunctions from Materialised Faceted Taxonomies: Algorithms and Complexity". *Knowledge and Information Systems (KAIS)*, 9(4):430–467, 2006.

[37] Yannis Tzitzikas, Anastasia Analyti, and Nicolas Spyratos. "Compound Term Composition Algebra: The Semantics". *LNCS Journal on Data Semantics*, 2:58–84, 2005.

[38] Yannis Tzitzikas, Anastasia Analyti, Nicolas Spyratos, and Panos Constantopoulos. "An Algebraic Approach for Specifying Compound Terms in Faceted Taxonomies". In *Information Modelling and Knowledge Bases XV, 13th European-Japanese Conference on Information Modelling and Knowledge Bases, EJC'03*, pages 67–87. IOS Press, 2004.

[39] Yannis Tzitzikas, Raimo Launonen, Mika Hakkarainen, Pekka Kohonen, Tero Leppanen, Esko Simpanen, Hannu Tornroos, Pekka Uusitalo, and Pentti Vanska. "FASTAXON: A system for FAST (and Faceted) TAXONomy design.". In *Proceedings of 23th Int. Conf. on Conceptual Modeling, ER'2004*, pages 841–843, Shanghai, China, November 2004. (an on-line demo is available at http://fastaxon.erve.vtt.fi/).

[40] Yannis Tzitzikas and Carlo Meghini. "Ostensive Automatic Schema Mapping for Taxonomy-based Peer-to-Peer Systems". In *Seventh International Workshop on Cooperative Information Agents, CIA-2003*, pages 78–92, Helsinki, Finland, August 2003. (Best Paper Award).

[41] Yannis Tzitzikas and Carlo Meghini. "Query Evaluation in Peer-to-Peer Networks of Taxonomy-based Sources". In *Procs. of 19th Int. Conf. on Cooperative Information Systems, CoopIS'2003*, pages 263–281, Catania, Sicily, Italy, November 2003.

[42] Yannis Tzitzikas, Nicolas Spyratos, and Panos Constantopoulos. "Mediators over Taxonomy-based Information Sources". *VLDB Journal*, 14(1):112–136, 2005.

[43] Yannis T. Tzitzikas. *"Collaborative Ontology-based Information Indexing and Retrieval"*. PhD thesis, Department of Computer Science - University of Crete, September 2002.

[44] B. C. Vickery. "Knowledge Representation: A Brief Review". *Journal of Documentation*, 42(3):145–159, 1986.

[45] W. A. Woods. "Understanding Subsumption and Taxonomy". In *Principles of Semantic Networks*, chapter 1. Morgan Kaufmann Publishers, 1991.

[46] K. Yee, K. Swearingen, K. Li, and M. Hearst. "Faceted Metadata for Image Search and Browsing". In *Proceedings of the Conf. on Human Factors in Computing Systems (CHI'03)*, pages 401–408, April 2003.

Appendix A: Proofs

In this Appendix, we give the proofs of the propositions appearing in the paper. Proofs that follow immediately from Definitions are omitted. We first prove an auxiliary lemma.

Lemma 1 Let the compound terminologies S_i, $i = 1, ..., n$, be *o-filters*. Then, $\oplus(S_1, ..., S_n)$ is an *o-filter*.

Proof:

Let $S = S_1 \oplus ... \oplus S_n$ such that $Br(S_i) = S_i$, for $i = 1, ..., n$. We will show $Br(S) = S$. Obviously, $S \subseteq Br(S)$. Let $s \in Br(S_1 \oplus ... \oplus S_n)$. Then, $\exists s' \in S_1 \oplus ... \oplus S_n$ such that $s' \leq s$. Let $s' = s'_1 \cup ... \cup s'_n$, where $s'_i \in S_i$, for $i = 1, ..., n$. Thus, $s = s_1 \cup ... \cup s_n$, where $s_i \in S_i$ and $s'_i \leq s_i$, for $i = 1, ..., n$. Thus, $s_i \in Br(S_i)$, for $i = 1, ..., n$. Therefore, $s \in Br(S_1) \oplus ... \oplus Br(S_n) = S_1 \oplus ... \oplus S_n$. Thus, $Br(S) = S$. $\diamond$

Prop. 3.1 Let S be a compound terminology over a terminology $\mathcal{T}$, and let s and s' be two elements of S. It holds: $s \preceq s'$ iff $\hat{I}(s) \subseteq \hat{I}(s')$ in every model I of $(\mathcal{T}, \leq)$.

Proof:

($\Rightarrow$)

Let $s = \{a_1, ..., a_m\}$ and $s' = \{b_1, ..., b_n\}$. If $s \preceq s'$, then for each b_i, $i = 1, ..., n$, there exists an a_j, $j = 1, ..., m$, such that $a_j \leq b_i$. This means that in every model I of $(\mathcal{T}, \leq)$, it holds $\hat{I}(a_j) \subseteq \hat{I}(b_i)$.

Now, as $\hat{I}(s) = I(a_1) \cap ... \cap I(a_m)$ and $\hat{I}(s') = I(b_1) \cap ... \cap I(b_n)$, it is evident that it holds $\hat{I}(s) \subseteq \hat{I}(s')$, in every model I of $(\mathcal{T}, \leq)$.
($\Leftarrow$)
Let $s = \{a_1, ..., a_m\}$ and $s' = \{b_1, ..., b_n\}$. As it has been shown in [43], $\hat{I}(s) \subseteq \hat{I}(s')$ in every model I of $(\mathcal{T}, \leq)$ iff $r(s \cup s') = r(s)$, where $r(\{t_1, ..., t_k\}) = minimal_{\leq}(\{c(t_1), ..., c(t_k)\})$ where $c(t)$ denotes the equivalence class of a term t. However, $r(s \cup s') = r(s)$ can hold only if for each $t' \in s'$ exists $t \in s$ such that $t \leq t'$. Thus, it must hold $s \preceq s'$. $\diamond$

Prop. 4.1 Let the compound terminologies S_i, $i = 1, ..., n$, and let $P \subseteq G_{S_1,...,S_n}$. It holds:

1. if S_i, $i = 1, ..., n$, are *o-filters* and $P = G_{S_1,...,S_n}$ then $\oplus_P(S_1, ..., S_n) = S_1 \oplus ... \oplus S_n$,
2. if $P = \emptyset$ then $\oplus_P(S_1, ..., S_n) = \bigcup_{i=1}^{n} S_i$,
3. if S_i, $i = 1, ..., n$, are *o-filters* then $\oplus_P(S_1, ..., S_n)$ is an *o-filter*, and
4. $\oplus_P(S_1, ..., S_n) = \oplus_P(S_{l_1}, ..., S_{l_n})$, where $(l_1, ..., l_n)$ is a permutation of $\{1, ..., n\}$. (*Commutativity*)

Proof:
1. For simplification, we will prove that the statement holds for n=2. That is, we will show that $S \oplus_{G_{S,S'}} S' = S \oplus S'$. The general proof goes similarly. It holds $S \oplus_{G_{S,S'}} S' = S \cup S' \cup Br(S \oplus S' - S - S')$.
We will first show that $S \cup S' \cup Br(S \oplus S' - S - S') \subseteq S \oplus S'$. From Lemma 1, it follows that $Br(S \oplus S') = S \oplus S'$. Therefore, $S \cup S' \cup Br(S \oplus S' - S - S') \subseteq S \cup S' \cup S \oplus S' = S \oplus S'$.
We will now show that $S \oplus S' \subseteq S \cup S' \cup Br(S \oplus S' - S - S')$. Let $s \in S \oplus S'$. If $s \in S \cup S'$ then obviously $s \in S \cup S' \cup Br(S \oplus S' - S - S')$. Otherwise, $s \in S \oplus S' - S - S' \subseteq Br(S \oplus S' - S - S')$, and thus $s \in S \cup S' \cup Br(S \oplus S' - S - S')$.
Therefore, it holds $S \oplus_{G_{S,S'}} S' = S \oplus S'$.

2. It follows immediately from the definition of plus-product and the fact that $Br(P) = \emptyset$, for $P = \emptyset$.

3. Let $S = \oplus_P(S_1, ..., S_n)$, such that $Br(S_i) = S_i$, for $i = 1, ..., n$. It holds $Br(S) = Br(S_1) \cup ... \cup Br(S_n) \cup Br(Br(P)) = S_1 \cup ... \cup S_n \cup Br(P) = S$.

4. It follows immediately from the fact that $S_1 \cup ... \cup S_n = S_{l_1} \cup ... \cup S_{l_n}$. $\diamond$

Prop. 4.4 $\oplus_P(S_1, ..., S_n) = \oplus_{P'}(S_1, ..., S_n)$ iff $minimal_{\preceq}(P) = minimal_{\preceq}(P')$.

Proof:
It follows immediately from the fact that $Br(P) = Br(P')$ iff $minimal_{\preceq}(P) = minimal_{\preceq}(P')$. $\diamond$

Prop. 4.5 Let the compound terminologies S_1, S_2, and S_3 be *o-filters*. It holds:

$$(S_1 \oplus_{P_1} S_2) \oplus_{P_2} S_3 = S_1 \oplus_{P_1'} (S_2 \oplus_{P_2'} S_3), \text{ where}$$
$$P_1' = P_1 \cup (P_2 - S_2 \oplus S_3),$$
$$P_2' = \{s_2 \cup s_3 \mid \exists s \in P_2,\ \emptyset \neq s_2, s_3 \subseteq s,\ s_2 \in S_2, s_3 \in S_3\}.$$

Proof:
First, we will show that P_1', P_2' are well-defined. In particular, we will show that

(i) $P_1' \subseteq S_1 \oplus (S_2 \oplus_{P_2'} S_3) - S_1 - S_2 \oplus_{P_2'} S_3$, (ii) $P_2' \subseteq S_2 \oplus S_3 - S_2 - S_3$

First, we will show (i). From Definition, it holds $P_1 \subseteq S_1 \oplus S_2 - S_1 - S_2$. It holds $S_1 \oplus S_2 - S_1 - S_2 \subseteq S_1 \oplus (S_2 \oplus_{P_2'} S_3) - S_1 - S_2 \oplus_{P_2'} S_3$. Therefore, $P_1 \subseteq S_1 \oplus (S_2 \oplus_{P_2'} S_3) - S_1 - S_2 \oplus_{P_2'} S_3$.
From Definition, it holds $P_2 \subseteq (S_1 \oplus_{P_1} S_2) \oplus S_3 - S_1 \oplus_{P_1} S_2 - S_3$. As $S_1 \subseteq S_1 \oplus_{P_1} S_2$, it holds $P_2 \cap S_1 = \emptyset$. Obviously, $P_2 \subseteq S_1 \oplus S_2 \oplus S_3$. From this and the definition of P_2', it follows that

$P_2 \subseteq S_1 \oplus (S_2 \oplus_{P_2'} S_3)$. From this and the facts that $P_2 \cap S_1 = \emptyset$ and $S_2 \oplus_{P_2'} S_3 \subseteq S_2 \oplus S_3$ (note that S_2, S_3 are *o-filters*), it follows that $P_2 - S_2 \oplus S_3 \subseteq S_1 \oplus (S_2 \oplus_{P_2'} S_3) - S_1 - S_2 \oplus_{P_2'} S_3$. Thus, from the definition of P_1', (i) holds.

Now we will show (ii). From Definition, $P_2 \subseteq (S_1 \oplus_{P_1} S_2) \oplus_{P_2} S_3 - S_1 \oplus_{P_1} S_2 - S_3 \subseteq (S_1 \oplus S_2) \oplus S_3 - S_1 - S_2 - S_3$. From the definition of P_2', it follows $P_2' \subseteq S_2 \oplus S_3 - S_2 - S_3$. Thus (ii) holds.

We will now show that $(S_1 \oplus_{P_1} S_2) \oplus_{P_2} S_3 \subseteq S_1 \oplus_{P_1'} (S_2 \oplus_{P_2'} S_3)$.
Let $s \in (S_1 \oplus_{P_1} S_2) \oplus_{P_2} S_3$.
If $s \in S_1 \oplus S_2$ then $s \in S_1 \oplus_{P_1'} (S_2 \oplus_{P_2'} S_3)$ as $P_1 \subseteq P_1'$.
If $s \in S_1 \oplus S_3$ then $s \in S_1 \oplus_{P_1'} (S_2 \oplus_{P_2'} S_3)$ as $P_2 - S_2 \oplus S_3 \subseteq P_1'$.
If $s \in S_2 \oplus S_3$ then from the definition of P_2', it follows that $s \in S_1 \oplus_{P_1'} (S_2 \oplus_{P_2'} S_3)$.
If $s \in S_1 \oplus S_2 \oplus S_3 - S_1 \oplus S_2 - S_1 \oplus S_3 - S_2 \oplus S_3$ then $s \in S_1 \oplus_{P_1'} (S_2 \oplus_{P_2'} S_3)$, as $P_2 - S_2 \oplus S_3 \subseteq P_1'$.

We will now show that $S_1 \oplus_{P_1'} (S_2 \oplus_{P_2'} S_3) \subseteq (S_1 \oplus_{P_1} S_2) \oplus_{P_2} S_3$.
Let $s \in S_1 \oplus_{P_1'} (S_2 \oplus_{P_2'} S_3)$.
If $s \in S_1 \oplus S_2$ then $s \in (S_1 \oplus_{P_1} S_2) \oplus_{P_2} S_3$, as $P_1' \cap (S_1 \oplus S_2) = P_1$.
If $s \in S_1 \oplus S_3$ then $s \in (S_1 \oplus_{P_1} S_2) \oplus_{P_2} S_3$, as $P_1' \cap (S_1 \oplus S_3) \subseteq P_2$.
If $s \in S_2 \oplus S_3$ then from the definition of P_2', it follows that $s \in (S_1 \oplus_{P_1} S_2) \oplus_{P_2} S_3$.
If $s \in S_1 \oplus S_2 \oplus S_3 - S_1 \oplus S_2 - S_1 \oplus S_3 - S_2 \oplus S_3$ then $s \in (S_1 \oplus_{P_1} S_2) \oplus_{P_2} S_3$, as $P_1' - S_1 \oplus S_2 \subseteq P_2$.

Thus, $(S_1 \oplus_{P_1} S_2) \oplus_{P_2} S_3 = (S_1 \oplus_{P_1'} S_2) \oplus_{P_2'} S_3$. $\diamond$

Prop. 4.6 Let the compound terminologies S_i, $i = 1, ..., n$, and let $N \subseteq G_{S_1,...,S_n}$. It holds:

1. if S_i, $i = 1, ..., n$, are separated and $N = G_{S_1,...,S_n}$ then $\ominus_N(S_1, ..., S_n) = \bigcup_{i=1}^n S_i$,
2. if $N = \emptyset$ then $\ominus_N(S_1, ..., S_n) = S_1 \oplus ... \oplus S_n$,
3. if S_i, $i = 1, ..., n$, are *o-filters* then $\ominus_N(S_1, ..., S_n)$ is an *o-filter*, and
4. $\ominus_N(S_1, ..., S_n) = \ominus_N(S_{l_1}, ..., S_{l_n})$, where $(l_1, ..., l_n)$ is a permutation of $\{1, ..., n\}$. (*Commutativity*)

Proof:
1. For simplification, we will prove this statement for n=2. That is, we will show that $S \ominus_{G_{S,S'}} S' = S \cup S'$. The general proof goes similarly. It holds $S \ominus_{G_{S,S'}} S' = S \oplus S' - Nr(S \oplus S' - S - S')$. Thus, it is enough to show that $S \oplus S' - Nr(S \oplus S' - S - S') = S \cup S'$.
We will show that $S \cup S' \subseteq S \oplus S' - Nr(S \oplus S' - S - S')$. Let $s \in S \cup S'$. We will show that $s \notin Nr(S \oplus S' - S - S')$. If $s \in Nr(S \oplus S' - S - S')$ then $\exists s' \in S \oplus S' - S - S'$ such that $s \preceq s'$. Let $s' = s_1 \cup s_2$ s.t. $s_1 \in S$ and $s_2 \in S'$. Obviously, $s_1 \neq \emptyset$ and $s_2 \neq \emptyset$. However, if $s \in S$ then since S, S' are separated, it is not possible that $s \preceq s_2$, and thus it is not possible that $s \preceq s'$. If $s \in S'$ then since S, S' are separated, it is not possible that $s \preceq s_1$, and thus it is not possible that $s \preceq s'$. Therefore, it is not possible that $s \in Nr(S \oplus S' - S - S')$, and thus $s \in S \oplus S' - Nr(S \oplus S - S - S')$.

We will show that $S \oplus S' - Nr(S \oplus S' - S - S') \subseteq S \cup S'$. Let $s \in S \oplus S' - Nr(S \oplus S' - S - S')$. Assume that $s \notin S \cup S'$. Then, $s = s_1 \cup s_2$ such that $s_1 \in S$, $s_2 \in S'$, and $s_1, s_2 \neq \emptyset$. But then $s \in S \oplus S' - S - S' \subseteq Nr(S \oplus S' - S - S')$, which is impossible. Therefore, $s \in S \cup S'$.

2. It follows immediately from the definition of minus-product and the fact that $Nr(N) = \emptyset$, for $N = \emptyset$.

3. Let $S = \ominus_N(S_1, ..., S_n)$ such that $Br(S_i) = S_i$, for $i = 1, ..., n$. Obviously, $S \subseteq Br(S)$. We will show that $Br(S) \subseteq S$. It holds $Br(S) = Br(S_1 \oplus ... \oplus S_n - Nr(N))$. We will show $Br(S_1 \oplus ... \oplus S_n - Nr(N)) \subseteq S_1 \oplus ... \oplus S_n - Nr(N)$. Let $s \in Br(S_1 \oplus ... \oplus S_n - Nr(N))$. Then, there is $s' \in S_1 \oplus ... \oplus S_n - Nr(N)$ such

that $s' \preceq s$. If $s \in Nr(N)$ then $s' \in Nr(N)$, which is impossible. Thus, $s \in Br(S_1 \oplus ... \oplus S_n) - Nr(N)$. From Lemma 1, it now follows that $S_1 \oplus ... \oplus S_n - Nr(N)$. Therefore, $Br(S) = S$.

4. It follows immediately from the fact that $S_1 \oplus ... \oplus S_n = S_{l_1} \oplus ... \oplus S_{l_n}$. $\diamond$

Prop. 4.7 Let the compound terminologies S_i, for $i = 1, ..., n$, be separated and let $l \leq n-2$. It holds:

1. $\ominus_{N_1}(S_1, ..., S_{n-1}) \ominus_{N_2} S_n = \ominus_{N_1 \cup N_2}(S_1, ..., S_n)$.
2. $(\ominus_{N_1}(S_1, ..., S_l)) \ominus_{N_2} (\ominus_{N_3}(S_{l+1}, ..., S_n)) = \ominus_{N_1 \cup N_2 \cup N_3}(S_1, ..., S_n)$.

Proof:
1. It holds $\ominus_{N_1}(S_1, ..., S_{n-1}) \ominus_{N_2} S_n = (S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n - Nr(N_2)$.
Additionally, it holds $\ominus_{N_1 \cup N_2}(S_1, ..., S_n) = S_1 \oplus ... \oplus S_n - Nr(N_1) - Nr(N_2)$.
Therefore, it is enough to show that $(S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n = S_1 \oplus ... \oplus S_n - Nr(N_1)$.
We will first show that $(S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n \subseteq S_1 \oplus ... \oplus S_n - Nr(N_1)$. Let $s \in (S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n$. We will show that $s \notin Nr(N_1)$. If $s \in S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)$ or $s \in S_n$ then obviously $s \notin Nr(N_1)$. Otherwise, $s = s_1 \cup s_2$, where $s_1 \in S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)$ and $s_2 \in S_n$ such that $s_1, s_2 \neq \emptyset$. Assume that $s \in Nr(N_1)$. Then, there is $s' \in N_1$ such that $s \preceq s'$. Since the terminologies $S_i, i = 1, ..., n$, are separated and $N_1 \subseteq G_{S_1,...,S_{n-1}}$, it follows that there are no t, t' such that $t \in s_2$, $t' \in s'$ and $t \leq t'$. From this, it follows that $s_1 \preceq s'$. Therefore, $s_1 \in Nr(N_1)$, which is impossible. Therefore, $s \notin Nr(N_1)$. Thus, $s \in S_1 \oplus ... \oplus S_n - Nr(N_1)$.

Let $s \in S_1 \oplus ... \oplus S_n - Nr(N_1)$. We will show that $s \in (S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n$. If $s \in S_1 \oplus ... \oplus S_{n-1}$ then obviously $s \in S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)$. Otherwise, $s = s_1 \cup s_2$, where $s_1 \in S_1 \oplus ... \oplus S_{n-1}$ and $s_2 \in S_n$ such that $s_1, s_2 \neq \emptyset$. If $s_1 \in Nr(N_1)$ then $s \in Nr(N_1)$, which is impossible. Therefore, $s_1 \notin Nr(N_1)$. Thus, $s \in (S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n$.

2. Similarly, to the above proof. $\diamond$

Prop. 4.10 Let the compound terminologies S_1, S_2, S_3 be separated. It holds:

$$(S_1 \ominus_{N_1} S_2) \ominus_{N_2} S_3 = S_1 \ominus_{N_1'} (S_2 \ominus_{N_2'} S_3), \text{ where}$$
$$N_1' = N_1 \cup (N_2 - S_2 \oplus S_3),$$
$$N_2' = N_2 \cap (S_2 \oplus S_3).$$

Proof:
First, we will show that N_1', N_2' are well-defined. In particular, we will show that
(i) $N_1' \subseteq S_1 \oplus (S_2 \ominus_{N_2'} S_3) - S_1 - S_2 \ominus_{N_2'} S_3$,
(ii) $N_2' \subseteq S_2 \oplus S_3 - S_2 - S_3$
First, we will show (i). From Definition, it holds $N_1 \subseteq S_1 \oplus S_2 - S_1 - S_2$. As $S_1 \oplus S_2 - S_1 - S_2 \subseteq S_1 \oplus (S_2 \ominus_{N_2'} S_3) - S_1 - S_2 \ominus_{N_2'} S_3$, it follows that $N_1 \subseteq S_1 \oplus (S_2 \ominus_{N_2'} S_3) - S_1 - S_2 \ominus_{N_2'} S_3$.
From Definition, $N_2 \subseteq (S_1 \ominus_{N_1} S_2) \oplus S_3 - S_1 \ominus_{N_1} S_2 - S_3$. As $S_1 \subseteq S_1 \ominus_{N_1} S_2$ (S_1, S_2 are separated), it holds $N_2 \cap S_1 = \emptyset$. Obviously, $N_2 \subseteq S_1 \oplus S_2 \oplus S_3$. From the definition of N_2', it follows that $N_2 \subseteq S_1 \oplus (S_2 \ominus_{N_2'} S_3)$. From this and the facts that $N_2 \cap S_1 = \emptyset$ and $S_2 \ominus_{N_2'} S_3 \subseteq S_2 \oplus S_3$, it follows that $N_2 - S_2 \oplus S_3 \subseteq S_1 \oplus (S_2 \ominus_{N_2'} S_3) - S_1 - S_2 \ominus_{N_2'} S_3$. Thus (i) holds.
Now we will show (ii). From Definition, it holds $N_2 \subseteq (S_1 \ominus_{N_1} S_2) \oplus S_3 - S_1 \ominus_{N_1} S_2 - S_3 \subseteq S_1 \oplus S_2 \oplus S_3 - S_1 - S_2 - S_3$. From the definition of N_2' it holds $N_2' \subseteq S_2 \oplus S_3 - S_2 - S_3$. Thus (ii) holds.

We will now show that $(S_1 \ominus_{N_1} S_2) \ominus_{N_2} S_3 \subseteq S_1 \ominus_{N_1'} (S_2 \ominus_{N_2'} S_3)$.
Let $s \in (S_1 \ominus_{N_1} S_2) \ominus_{N_2} S_3$.
If $s \in S_1 \oplus S_2$ then $s \in S_1 \ominus_{N_1'} (S_2 \ominus_{N_2'} S_3)$, as $N_1' \cap (S_1 \oplus S_2) = N_1$.
If $s \in S_1 \oplus S_3$ then $s \in S_1 \ominus_{N_1'} (S_2 \ominus_{N_2'} S_3)$, as $N_1' \cap (S_1 \oplus S_3) \subseteq N_2$.
If $s \in S_2 \oplus S_3$ then from the definition of N_2', it follows that $s \in S_1 \ominus_{N_1'} (S_2 \ominus_{N_2'} S_3)$.

If $s \in S_1 \oplus S_2 \oplus S_3 - S_1 \oplus S_2 - S_1 \oplus S_3 - S_2 \oplus S_3$ then $s \in S_1 \ominus_{N_1'} (S_2 \ominus_{N_2'} S_3)$, as $N_1' - S_1 \oplus S_2 \subseteq N_2$.

We will now show that $S_1 \ominus_{N_1'} (S_2 \ominus_{N_2'} S_3) \subseteq (S_1 \ominus_{N_1} S_2) \ominus_{N_2} S_3$.
Let $s \in S_1 \ominus_{N_1'} (S_2 \ominus_{N_2'} S_3)$.
If $s \in S_1 \oplus S_2$ then $s \in (S_1 \ominus_{N_1} S_2) \ominus_{N_2} S_3$, as $N_1 \subseteq N_1'$.
If $s \in S_1 \oplus S_3$ then $s \in (S_1 \ominus_{N_1} S_2) \ominus_{N_2} S_3$, as $N_2 - S_2 \oplus S_3 \subseteq N_1'$.
If $s \in S_2 \oplus S_3$ then from the definition of N_2', it follows that $s \in (S_1 \ominus_{N_1} S_2) \ominus_{N_2} S_3$.
If $s \in S_1 \oplus S_2 \oplus S_3 - S_1 \oplus S_2 - S_1 \oplus S_3 - S_2 \oplus S_3$ then $s \in (S_1 \ominus_{N_1} S_2) \ominus_{N_2} S_3$, as $N_2 - S_2 \oplus S_3 \subseteq N_1'$.

Thus, $(S_1 \ominus_{N_1} S_2) \ominus_{N_2} S_3 = S_1 \ominus_{N_1'} (S_2 \ominus_{N_2'} S_3)$. $\diamond$

Prop. 4.11 Let $\mathcal{T}_i$ be a basic compound terminology and $P, N \subseteq G_{T_i}$. Then, the compound terminologies T_i, $\overset{*}{\oplus}_P (T_i)$, and $\overset{*}{\ominus}_N (T_i)$ are *o-filters*.
Proof:
Obviously, $Br(T_i) = T_i$.
Let $S = \overset{*}{\oplus}_P T_i$, then $Br(S) = Br(T_i \cup Br(P)) = Br(T_i) \cup Br(Br(P)) = T_i \cup Br(P) = S$.
Let $S = \overset{*}{\ominus}_N T_i$, then $Br(S) = Br(\overset{*}{\oplus} T_i - Nr(N))$. We will show that $Br(\overset{*}{\oplus} T_i - Nr(N)) \subseteq \overset{*}{\oplus} T_i - Nr(N)$. Let $s \in Br(\overset{*}{\oplus} T_i - Nr(N))$. Then, there is $s' \in \overset{*}{\oplus} T_i - Nr(N)$ such that $s' \preceq s$. If $s \in Nr(N)$ then $s' \in Nr(N)$, which is impossible. Thus, $s \in \overset{*}{\oplus} T_i - Nr(N)$. Therefore, $Br(S) = S$. $\diamond$

Prop. 6.1 $IsValid(e, s)$ =TRUE iff $s \in S_e$
Proof:
It is evident that this holds because the algorithm just follows the definitions of the operations as given in Section 4.$\diamond$

Prop. 6.2 Let e be an expression and let $s \subseteq \mathcal{T}$. The complexity of $IsValid(e, s)$ is $O(|\mathcal{T}|^3 * |s| * |\mathcal{P} \cup \mathcal{N}|)$.
Proof:
Let $VT(e)$ be the time complexity of $IsValid(e, s)$. In the worst case, all of the following checks have to be made:

(i) $p \preceq s'$, $\forall p \in \mathcal{P}$, where s' is a constant $s' \subseteq s$.

(ii) $s' \preceq n$, $\forall n \in \mathcal{N}$, where s' is a constant $s' \subseteq s$.

(iii) $\{t\} \preceq s_i$, $\forall t \in \mathcal{T}_i$, $\forall i = 1, ..., k$, where k is the number of facets and s_i is a constant $s_i \subseteq s$.

We will now compute the complexity of the checks in (i) and (ii). Let s_1, s_2 be compound terms. Recall that $s_1 \preceq s_2$ iff $\forall t_2 \in s_2$, $\exists t_1 \in s_1$ s.t. $t_1 \leq t_2$. Thus, to check if $s_1 \preceq s_2$, the check $t_1 \leq t_2$ is made, at most for all $t_2 \in s_2$ and $t_1 \in s_1$. We will now compute the complexity of the check $t_1 \leq t_2$, where t_1, t_2 are constants. If we have stored only the transitive reduction of the subsumption relation $\leq$ then term subsumption reduces to the problem of *reachability* in the graph of the faceted taxonomy $\mathcal{F}$. Since at most $|\mathcal{T}|^2$ edges have to be processed, the time complexity of this problem is $O(|\mathcal{T}|^2)$ (for more see [22]). Thus, the complexity of the check $s_1 \preceq s_2$ is $O(|\mathcal{T}|^2 * |s_1| * |s_2|)$.
As we have $|\mathcal{P} \cup \mathcal{N}|$ goals of the form $p \leq s'$ and $s' \leq n$, where $p, n \in \mathcal{P} \cup \mathcal{N}$ and $|s'| \leq |s|$, the complexity of all checks in (i) and (ii) is $O(|\mathcal{T}|^3 * |s| * |\mathcal{P} \cup \mathcal{N}|)$.
Now we will compute the complexity of the checks in (iii). Suppose that i is fixed. As we have shown above, the check $\{t\} \preceq s_i$, for a term $t \in \mathcal{T}_i$ and $s_i \subseteq s$, takes $O(|\mathcal{T}|^2 * |s|)$ time. Thus, the complexity of the checks $\{t\} \preceq s_i$, $\forall t \in \mathcal{T}_i$, is $O(|\mathcal{T}|^3 * |s|)$. Therefore, the complexity of all checks in (iii) is $O(k * |\mathcal{T}|^3 * |s|)$.
In total, $VT(e) = O(|\mathcal{T}|^3 * |s| * |\mathcal{P} \cup \mathcal{N}| + k * |\mathcal{T}|^3 * |s|$. Assuming that in general $k \leq |\mathcal{P} \cup \mathcal{N}|$, it follows that: $VT(e) = O(|\mathcal{T}|^3 * |s| * |\mathcal{P} \cup \mathcal{N}|)$. $\diamond$

Appendix B: Navigation Tree

Algorithm .1 NavigationTree(e) (complete version)
Input: An expression e over $\mathcal{F}$
Output: A navigation tree (N, R)

```
subroutine CreateTree(n: Node)
    if Nm(n)="Top" then
        // Creating the initial nodes
        For each i = 1, ..., k
            n' := createNode( top(F_i), {top(F_i)}, top(F_i))
            AddChild(n, n')
            CreateTree(n')
        End For
    Else
        // Creating the children of a node on the basis of the focus term
        For each t ∈ Nar(Fc(n))
            Let s' := (s(n) − Fc(n)) ∪ {t}
            If IsValid(e, s') then
                n'=createNode(t, s', t)
                AddChild(n, n')
                CreateTree(n')

        // Creating the children of a node for "facet crossing"
        For each F_i ∈ F(e) − F(Fc(n))
            Let s_i := s(n)∩T_i
            If s_i = ∅ then
                Let s' := s(n) ∪ {top(F_i)}
                If IsValid(e, s') then
                    n'=createNode("by" + top(F_i), s', top(F_i) )
                    AddChild(n, n')
                    CreateTree(n')
            else
                If s_i = {t_i} and ∃t' ∈ Nar(t_i): IsValid( e, (s(n) − {t_i}) ∪ {t'}) then
                    n'=createNode("by"+top(F_i), s(n), t_i)
                    AddChild(n, n')
                    CreateTree(n')
        End For
End CreateTree


BEGIN
    topNode := createNode( "Top", "", "")        // creation of a dummy top node
    CreateTree(topNode)
    return topNode
END
```
