Towards practical taxonomic classification for description logics on the Semantic Web

Scott P. Sanner
Department of Computer Science
University of Toronto
Toronto, Ont, M5S 3H5, CANADA
ssanner@cs.toronto.edu

Abstract

Description logics offer a well-defined semantics for many common and useful reasoning tasks that can be formalized under the notion of subsumption and are currently finding use as a representation language for the Semantic Web (e.g. DAML+OIL). For this domain, description logics can be highly useful for reasoning about relationships between entities in distributed knowledge bases by classifying them into a taxonomic subsumption hierarchy. However, there are many practical considerations for designing a sound and complete, yet efficient algorithm for performing large-scale taxonomic classification. Traditionally, structural subsumption algorithms have provided efficient techniques for performing classification but have only supported relatively inexpressive languages. Consequently, in response to the need for efficient taxonomic classification of more expressive languages, we develop an approach that extends previous structural subsumption algorithms to support sound and complete classification of a moderately expressive description logic including both conjunctive and disjunctive constructors. Furthermore, we extend this algorithm to a more expressive description logic with the claim that its sources of incompleteness are infrequent and benign in practice. Finally, we show that for the expected distribution of concept structures, both of these taxonomic classification algorithms require polynomial time in the size of the knowledge base and argue that such a result is a necessity for practical classification algorithms that will scale with the expected growth of the Semantic Web.

1 Introduction

As languages for knowledge representation on the Semantic Web [Bemers-Lee *et al.*, 2001] and other media gain widespread acceptance, it is important that the associated reasoning tools have the ability to scale to the unprecedented quantities of structured content that are likely to emerge. Furthermore, it is important that the majority of this reasoning be automated since direct human involvement is both costly and time-consuming. Thus, the desire for automated reasoning coupled with the expected growth rate of the Semantic Web poses a set of constraints on reasoning algorithms whose sheer magnitude has not been encountered previously in the field of knowledge representation and reasoning.

Consequently, we examine practical applications of reasoning on the Semantic Web that can be formalized under the notion of description logic subsumption. Furthermore, we analyze the constraints that this domain places on such reasoning in an attempt to provide an efficient, scalable, and provably correct algorithm to satisfy our reasoning needs within these constraints.

1.1 Description logics and the Semantic Web

As the Semantic Web comes of age, it is likely that there will be a shift from the database-like specification of knowledge bases (KB's) containing primitive concepts (e.g. a list of a retailer's products) to KB's containing logical compositions of primitive objects collected from distributed KB's. Consequently, the choice of a description logic in the spirit of KL-ONE [Woods and Schmolze, 1992] such as DAML+OIL [Horrocks *et al.*, 2001] as a representation language for the Semantic Web is well-suited to this domain since it automates reasoning about relationships between such composed concepts.

To motivate our discussion, we use an example drawn from online retail sales: If three online office supply retailers list their products in DAML+OIL, each could specify the following primitive concept URI's for their pencil products:

- *Retailer-1*'s primitive pencil concept:
  $http://www.ret1.com/pg.daml\#Ret1\_Pencil$
- *Retailer-2*'s primitive pencil concept:
  $http://www.ret2.com/pg.daml\#Ret2\_Pencil$
- *Retailer-3*'s primitive pencil concept:
  $http://www.ret3.com/pg.daml\#Ret3\_Pencil$

It is important to note that each of these concepts is unique and primitive (i.e. lacking definitions of sufficient conditions for membership) and thus have no intrinsic relation to each other outside of natural language interpretation.

Now, it is expected that other organizations wanting to construct their own product ontologies will define composite pencil concepts referring to the set of pencils that agree with their

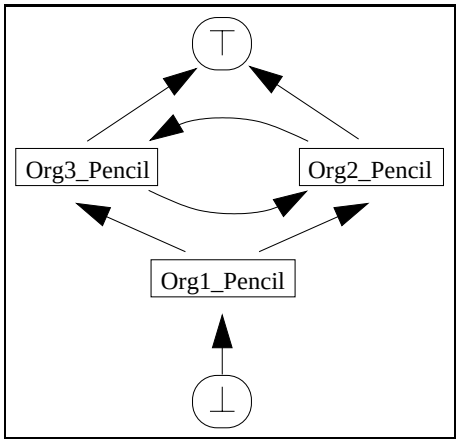

Figure 1: A taxonomy of defined concepts.

individual organizational definition and needs. For example: [1]

- *Organization-1*'s pencil concept definition:
  $Org1\_Pencil \doteq Ret1\_Pencil \sqcup Ret2\_Pencil$
- *Organization-2*'s pencil concept definition:
  $Org2\_Pencil \doteq Ret1\_Pencil \sqcup Ret2\_Pencil \sqcup Ret3\_Pencil$
- *Organization-3*'s pencil concept definition:
  $Org3\_Pencil \doteq Org1\_Pencil \sqcup Ret3\_Pencil$

Note that these organizations have defined their pencil concept in terms of disjunctions of other primitive and composite concepts defined by the various retailers.[2] It is this *logical compositional definition from common primitives* that makes it possible to reason about the relationships between such concepts defined in distributed and independently developed knowledge bases.[3]

To demonstrate the relationships that can be inferred between the above defined concepts, we have drawn a sample taxonomy in Figure 1. Arrows represent a subsumption (a.k.a. *kind-of*) relation that points from the subsumed concept to the subsuming concept. The special concept $\top$ represents the universal concept which subsumes every concept and the special concept $\bot$ represents the empty concept that is subsumed by every other concept.[4]

Such an automatically induced taxonomic structure induces a subsumption hierarchy which has two main properties: 1) It is minimal in the sense that no links are redundant, and 2) It is complete for the reflexive/transitive closure of all subsumption relations (i.e. no subsumption relation is missing). Thus, we can deduce by the mutual subsumption relation between *Org2_Pencil* and *Org3_Pencil* that these concepts are equivalent and from their subsumer relation to *Org1_Pencil* that both are more general than this concept. This latter subsumption relationship implies that any instance of *Org1_Pencil* must also necessarily be an instance of both *Org2_Pencil* and *Org3_Pencil*.

Note that while these relationships can be provably derived from the concept definitions, all relationships are not immediately apparent from the definitions. Yet these inferences can be extremely important since *Organization-2* can now determine that both *Organization-3*'s and *Organization-1*'s pencil definitions are equivalent and can be substituted for each other. Furthermore, any pencil that may be inferred to satisfy *Organization-1*'s definition will also necessarily satisfy the definition of *Organization-1* and *Organization-3*.

Note that it would be a mistake to define axioms stating necessary relationships between concepts as opposed to using their full definitions to automate this inference (e.g. using *Ret1_Pencil* $\sqsubseteq$ *Org1_Pencil* and *Ret2_Pencil* $\sqsubseteq$ *Org1_Pencil* in place of the current disjunctive definition of *Org1_Pencil*). Such a definition does not capture the fact that an instance of *Org1_Pencil* is necessarily either a *Ret1_Pencil* or a *Ret2_Pencil* and thus it is impossible to make the inferred relationships in Figure 1 with such an incomplete definition.

Consequently, it is the purpose of this paper to describe an efficient and practical algorithm for reasoning about the subsumption relationships between defined concepts that are composed from a set of logical constructors. This is one of the primary goals of the field of description logics and next we will focus on some limitations of current approaches.

1.2 Problems with current approaches

Current research in description logic classification has focused on a on worst-case analysis for sound and complete subsumption reasoning with respect to an extensional (i.e. model-theoretic) semantics by providing satisfiability based algorithms for subsumption testing [Horrocks *et al.*, 2000]. Unfortunately, such reasoning is intrinsically NP-Complete even for relatively inexpressive languages [Levesque and Brachman, 1985]. Furthermore, this approach has focused on the tractability of an individual subsumption test as opposed to the tractability of constructing a taxonomy.

This approach has been criticized by Woods [1991], who states that the "primary tractability concern is not the cost of subsumption, but the cost of classifying into a large taxonomy". And this criticism is especially relevant for the application discussed here since we will need an algorithm that can classify the massive amounts of content available on the Semantic Web.

As pointed out by Woods, structural subsumption approaches lend themselves to efficient techniques for taxonomic classification by reusing redundant information over multiple subsumption tests in order to amortize the cost of subsumption testing. However, structural subsumption techniques use an intensional definition of subsumption that is not

[1] From here on out, we will assume that each retailer or organization has a unique namespace and thus we will drop the namespace from the concept URI. Consequently, $http://www.ret1.com/pg.daml\#Ret1\_Pencil$ will simply be written as $Ret1\_Pencil$.

[2] See Table 1 for the definition of the description logic symbols used here.

[3] As a side note, the use of description logics for describing compositions should also be instrumental in moving the web away from a database-like representation where mappings need to be explicitly specified to a paradigm where such mappings can be automatically inferred.

[4] The importance of including these root $\top$ and $\bot$ concepts will become apparent once we start to define algorithms for performing concept classification.

necessarily extensionally complete. While the use of structural subsumption has yielded provably efficient classification algorithms (e.g. Classic [Borgida *et al.*, 1989]) it has traditionally been limited due to its inability to provide extensionally complete reasoning for expressive languages as well as its inability to support many commonly used combinations of description logic constructors [Brachman *et al.*, 1991]. Specifically, for this application, both conjunction and disjunction are extremely important constructors for reasoning about logically composed concepts, but both constructors have not been jointly addressed in any previous work on provably complete structural subsumption techniques for classification.

1.3 A potential solution

Before we provide a potential solution to this problem, let us first define the constraints on description logic classification posed by the Semantic Web:

1. The taxonomic classification algorithm must run in polynomial-time in the size of the taxonomy.[5]
2. The language must be expressive enough to build practically useful composite concepts. Thus, we need at least conjunction, disjunction, and commonly used role restrictions such as existential, universal, and qualified cardinality restrictions.[6]

With these requirements however, it seems we are at a bit of an impasse: The extensional approaches to subsumption do not place their primary focus on efficient taxonomic classification and it is theoretically impossible for them to achieve polynomial time complexity if complete subsumption algorithms are used. And although the intensional approaches can provide polynomial-time taxonomic classification algorithms, they do not seem to support expressive enough languages for our purposes.

Consequently, a solution satisfying both of the above constraints may need to sacrifice requirements of extensional soundness and completeness or alternately show that the super-polynomial classification cases do not occur for the expected distribution of concepts in our domain. Since there are multiple sources of super-polynomial complexity for expressive languages, our solution will utilize a combination of these approaches: sacrificing completeness in some cases where it is arguably benign to avoid exponential blow-up, and making the claim that for the other cases, exponential blow-up is not likely to occur for the expected class of concept structures in our domain.

Thus, in order to achieve the above goals, we build on the established efficiency of current structural classification algorithms and augment them with the ability to support both conjunctive and disjunctive constructors. For a moderately expressive language, we find that our selection of constructors yields a polynomial-time extensionally sound and complete algorithm when applied to the expected distribution of concept structures. For more expressive languages however, we sacrifice extensional completeness with the claim that the lost subsumptions are benign for most large-scale practical reasoning applications on the Semantic Web.

Ultimately, our claim of achieving the above goals is predicated upon the accuracy of our tradeoff assumptions. However we believe these claims are well-grounded in practical applications and in some sense, given the impossibility of extensionally complete, polynomial-time taxonomic classification algorithms for expressive description logics, these results are in some sense the best that can be hoped for.

[5] We assume any time complexity beyond polynomial to be intractable when applied to the Semantic Web.

[6] We exclude complement here since composition of concepts is expected to occur mainly through conjunction, disjunction, and role restrictions.

2 Language and semantics

Having previously discussed the language elements that are useful in a compositional domain such as the Semantic Web, let us now define the language elements and their model-theoretic semantics.

2.1 Language definition

Table 1 lists the constructor, definitional, and axiomatic elements for two languages, $\mathcal{L}_1$ and $\mathcal{L}_2$. Both languages are a subset of the DAML+OIL language used for the Semantic Web. With respect to DAML+OIL, $\mathcal{L}_2$ lacks complement; transitive, functional, and inverse roles; role restrictions involving individuals or set selection; and disjointness assertions. However, we consider the subsets $\mathcal{L}_1$ and $\mathcal{L}_2$ since they still capture some of the most useful constructors for logically composed concepts.

2.2 Semantics

We provide a model theoretic semantics for $\mathcal{L}_1$ and $\mathcal{L}_2$ in Table 1. Under an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, $\Delta^{\mathcal{I}}$ is a nonempty domain, and an interpretation function $\cdot^{\mathcal{I}}$ maps from concept names into a subset of $\Delta^{\mathcal{I}}$ and from role names into a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

2.3 Additional language restrictions

Since we will be using structural subsumption for efficient taxonomic classification, we must make a few additional restrictions on the language to make it amenable to this approach. These three restrictions involve the following:

1. We must prevent any non-primitive concept from referencing itself or a subclass in a definition (including through a role restriction). This restriction is not limiting since $Person \sqsubseteq \forall child.Person$ is still legal and there are few non-primitive concepts that would require such a recursive definition.
2. We limit axioms to primitive concepts only. However this is not as limiting as it sounds since there are semantically equivalent rewrites of some axioms involving one primitive concept and one structured concept. For example, for primitive concepts C, D, and E, $C \sqsubseteq D$ and $C \sqsubseteq E$ will allow $C \sqsubseteq D \sqcap E$ to be inferred via structural subsumption. And likewise, $D \sqsubseteq C$ and $E \sqsubseteq C$ will allow $D \sqcup E \sqsubseteq C$ to be inferred as well. Thus, axioms from a primitive concept to a disjunctive child or from a primitive concept to a conjunctive parent can be defined in this manner.

Languages $\mathcal{L}_1$ and $\mathcal{L}_2$

Constructor	*Syntax*	*Semantics*
Concept name	C	$C^{\mathcal{I}}$ (where $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$)
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Existential restriction	$\exists R.C$	$\{x \mid \exists y : R^{\mathcal{I}}(x,y) \wedge C^{\mathcal{I}}\}$
Min cardinality restriction	$\geqslant nR$	$\{x \mid \sharp\{y \vert R^{\mathcal{I}}(x,y)\} \geq n\}$
Qualified min cardinality restriction	$\geqslant nR.C$	$\{x \mid \sharp\{y \vert R^{\mathcal{I}}(x,y) \wedge C^{\mathcal{I}}(y)\} \geq n\}$
Role name	R	$R^{\mathcal{I}}$ (where $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$)

Definitional or Axiomatic Constraint	*Syntax*	*Semantic Constraint*
Concept definition	$C \doteq D$	$C^{\mathcal{I}} \equiv D^{\mathcal{I}}$
Concept subsumption axiom	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Role subsumption axiom	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

Language $\mathcal{L}_2$ only

Constructor	*Syntax*	*Semantics*
Universal (value) restriction	$\forall R.C$	$\{x \mid \forall y : R^{\mathcal{I}}(x,y) \rightarrow C^{\mathcal{I}}\}$
Max cardinality restriction	$\leqslant nR$	$\{x \mid \sharp\{y \vert R^{\mathcal{I}}(x,y)\} \leq n\}$
Qualified max cardinality restriction	$\leqslant nR.C$	$\{x \mid \sharp\{y \vert R^{\mathcal{I}}(x,y) \wedge C^{\mathcal{I}}(y)\} \leq n\}$

Table 1: Extensional Semantics for $\mathcal{L}_1$ and $\mathcal{L}_2$

3. We limit concepts to have at most one equivalence definition. For example, $A \doteq B \sqcap (C \sqcup D)$ is ok, but one could not add an additional definition such as $A \doteq E \sqcup F$ to a knowledge base already containing the first definition.

3 Subsumption definition

Now that we have defined the semantics for our language, we will proceed to give two definitions for subsumption in these languages. The first definition is extensionally complete via its direct reference to the model-theoretic semantics while the second definition is complete for $\mathcal{L}_1$ but not for $\mathcal{L}_2$.

3.1 Extensional subsumption

Under an extensional approach to subsumption for $C \sqsubseteq D$, we are effectively checking whether every model of C also satisfies D. To do this, we can build the concept $C \sqcap \neg D$, and determine whether any models could satisfy it. If we can show $C \sqcap \neg D$ is unsatisfiable, then whenever C holds for a model, clearly D must also hold. This effectively reduces subsumption testing to an unsatisfiability test and yields extensionally sound and complete results for both $\mathcal{L}_1$ and $\mathcal{L}_2$. Many NP-Complete but nevertheless relevantly efficient subsumption algorithms have been developed using this method (e.g. [Horrocks *et al.*, 2000]).

As pointed out by Donini [2002], it is interesting to note that in constructing $C \sqcap \neg D$, we are implicitly including complement in the language that we are using for subsumption inference despite the fact that we stated our concept within the constraints of a less expressive language. Consequently, it is the case that the language for which we are performing an unsatisfiability test is more expressive than the original language in which we stated our subsumption test. Thus, although we present no formal results, it may be the case that subsumption testing methods other than satisfiability can inherently perform more efficiently than the approach described here since an extensional subsumption approach would inherently require augmenting the langauge with the expressivity of complement.

3.2 Intensional subsumption

While the previous method of subsumption inference can be considered proof via failure of an exhaustive search for a countermodel, intensional subsumption can be considered more of a direct theorem proving method based on formal logical rules for inference. Such rules for subsumption in $\mathcal{L}_1$ and $\mathcal{L}_2$ are given below in Table 2, however it should be noted that applying these subsumption rules directly to any two concept structures would *not* yield extensional completeness. For extensional completeness, a concept's structure must first be normalized before these rules are applied. However, since normalization and classification are interleaved processes, we postpone the discussion of normalization until the next section on classification.

Structural comparison of two concept structures [1,2,3]

Concept A type	*Concept B type*	*Intensional rule for $A \sqsubseteq_i B$ (i.e. A is intensionally subsumed by B)*
Primitive	Primitive	(Base) $A \sqsubseteq_i B$ iff B is in the reflexive transitive closure of $\sqsubseteq$ for A.
Primitive	Conjunctive	(Recursive) $A \sqsubseteq_i B$ iff for *every* conjunctive constituent B_j of B, $A \sqsubseteq_i B_j$.
Primitive	Disjunctive	(Recursive) $A \sqsubseteq_i B$ iff for *some* conjunctive constituent B_j of B, $A \sqsubseteq_i B_j$.
Conjunctive	Primitive	(Recursive) $A \sqsubseteq_i B$ iff for *some* conjunctive constituent A_i of A, $A_i \sqsubseteq_i B$.
Conjunctive	Conjunctive	(Recursive) $A \sqsubseteq_i B$ iff for *every* conjunctive constituent B_j in B, there is *some* conjunctive constituent A_i in A such that $A_i \sqsubseteq_i B_j$.
Conjunctive	Disjunctive	(Recursive) $A \sqsubseteq_i B$ iff for *some* conjunctive constituent A_i of A, there is some disjunctive constituent B_j of B such that $A_i \sqsubseteq_i B_j$.
Disjunctive	Primitive	(Recursive) $A \sqsubseteq_i B$ iff for *every* disjunctive constituent A_i of A, $A_i \sqsubseteq_i B$.
Disjunctive	Conjunctive	(Recursive) $A \sqsubseteq_i B$ iff for *every* disjunctive constituent A_i of A and *every* conjunctive constituent B_j of B, $A_i \sqsubseteq_i B_j$.
Disjunctive	Disjunctive	(Recursive) $A \sqsubseteq_i B$ iff for *every* disjunctive constituent A_i in A, there is *some* disjunctive constituent B_j in B such that $A_i \sqsubseteq_i B_j$.

Structural comparison of two role restriction structures [1,3,4]

Restriction A type	*Restriction B type*	*Intensional rule for $A \sqsubseteq_i B$ (i.e. A is intensionally subsumed by B)*
$\leqslant n_1 R_1.C_1$	$\leqslant n_2 R_2.C_2$	(Recursive) $A \sqsubseteq_i B$ iff $n_1 \leq n_2$ and $R_1 \sqsubseteq R_2$ and $C_1 \sqsubseteq_i C_2$.
$\geqslant n_1 R_1.C_1$	$\geqslant n_2 R_2.C_2$	(Recursive) $A \sqsubseteq_i B$ iff $n_1 \geq n_2$ and $R_1 \sqsubseteq R_2$ and $C_1 \sqsubseteq_i C_2$.
$\forall R_1.C_1$	$\forall R_2.C_2$	(Recursive) $A \sqsubseteq_i B$ iff $R_1 \sqsubseteq R_2$ and $C_1 \sqsubseteq_i C_2$.
Any other combination		(Base) $A \sqsubseteq_i B$ not possible, return *false*

Note 1: We introduce the symbol $\sqsubseteq_i$ to indicate an inferred intensional subsumption. For this definition, we use $\sqsubseteq$ to refer to asserted subsumptions such as axioms or subsumptions implicit in conjunctive and disjunctive definitions. For example, $A \doteq B \sqcap C$ implicitly contains $A \sqsubseteq B$ and $A \sqsubseteq C$ and $A \doteq B \sqcup C$ implicitly contains $B \sqsubseteq A$ and $C \sqsubseteq A$.

Note 2: One will note that the only possible concept types are primitive, disjunctive, and conjunctive when in fact, defined concepts can mix both conjunction and disjunction. However, this is not a problem since we build concepts in multiple stages with each stage being purely disjunctive or conjunctive and freely referring to the other type as a constituent. For example, $A \doteq B \sqcup (C \sqcap D)$ would be built as a named concept A with two disjunctive constituents: B and an anonymous conjunctive concept representing $C \sqcap D$. Also note that after DNF normlization, all concepts will be a single named disjunction of anonymous conjunctions of primitive concepts. See the normalization section for more information.

Note 3: Concepts cannot subsume individual role restrictions and vice versa except via direct definition, i.e. $A \doteq B \sqcup \leqslant nR.C$ implicitly contains $\leqslant nR.C \sqsubseteq A$ but other than such definitions, $\leqslant nR.C$ cannot subsume or be subsumed by any other concept. See the completeness proof for more information.

Note 4: We omit intensional subsumption rules for $\exists nR.C$, $\geqslant nR$, and $\leqslant nR$ since the first two can be converted to the form $\geqslant nR.C$ and the last can be converted to the form $\leqslant nR.C$. See the normalization section for more information.

Table 2: Intensional subsumption rules for $\mathcal{L}_1$ and $\mathcal{L}_2$

4 Algorithm definition

The classification algorithm seeks to take a new concept and insert it into the taxonomy such that the reflexive transitive closure of all taxonomy links yields all subsumption relationships (with respect to the subsumption definition) while ensuring that no links are redundant (i.e. the taxonomy is minimal). Due to the presence of mutual recursion between the normalization and classification algorithms, the algorithm can appear somewhat complex at first. Consequently, we will first discuss the following general steps for performing taxonomic classification of a concept:

1. *Normalize the concept description*: Given a target concept, we must first ensure that all equivalent concepts normalize to the same concept structure. If we skipped this step, it would be impossible to necessarily determine whether two concepts were equivalent. For example, $A \sqcap (B \sqcup C)$ and $(A \sqcap B) \sqcup (A \sqcap C)$ are equivalent, but this cannot be determined by purely structural rules until both are converted to disjunctive normal form, i.e. the form of the latter concept. For now it suffices to mention that normalization yields a concept in disjunctive normal form with all possible role restrictions merged and all redundant concept elements removed.
2. *Find the most specific subsumers (MSS)*: Given a target concept, find the set of all concepts in the taxonomy that subsume the target such that no concept in this set subsumes any other (i.e. discard subsumers that are not most specific since they can be inferred via reflexive transitive closure). Refer to this set as the mss set.
3. *Find the most general subsumees (MGS)*: Given a target concept, find the set of all concepts in the taxonomy subsumed by the target such that no concept in this set is subsumed by any other (i.e. again, the non-most general subsumees can be inferred via reflexive transitive closure). Refer to this set as the mgs set.
4. *Insert the concept into taxonomy*: Given a target concept and its mss and mgs sets, we now determine how to add the concept to the taxonomy to minimize the number of subsumption links and remove any redundant links if present.

Having defined the basic ideas behind taxonomic classification, we now formalize the algorithm in the form of a number of subprocedures given in Algorithms 1–5: $ClassifyKb()$, $Classify(target)$, $NormalizeAndDefine(target)$, $MSS(target)$, $MGS(target)$.

4.1 Classification of a knowledge base

In Algorithm 1 we first define the algorithm for classifying an entire knowledge base consisting of primitive concepts and relations, axioms, and concept and role restriction definitions. This algorithm is relatively straightforward, simply requiring that the taxonomy be initialized by inserting the root concepts $\top$ and $\bot$, the root relation $\top_{rel}$, all relations and primitive concepts, and any stated and default axioms relating them.[7]

[7] A concept or axiom is not known in the taxonomy until it has been added.

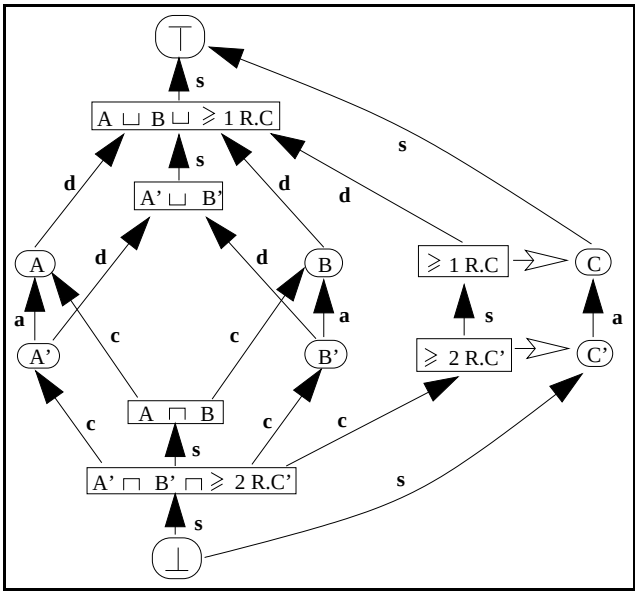

Figure 2: Example taxonomy showing different subsumption link types (closed arrow) as well as concept referents of restrictions (open arrow). Primitive concepts are displayed in rounded rectangles, defined concepts in normal rectangles. Note the different structural subsumptions and their conformance to the intensional subsumption definition.

After this, all structured concepts are inserted and classified into the taxonomy. Once this procedure has completed, the full subsumption taxonomy will have been built for the contents of the knowledge base.

It is first important to note that we use many different types of subsumption links in this algorithm. These link types follow:

- $\sqsubseteq_a$ – Any directly asserted, axiomatic subsumption.
- $\sqsubseteq_s$ – Any structural subsumption inference conforming to the intensional subsumption definition that has not already been stated as an axiom.
- $\sqsubseteq_c$ – A definitional link from a conjunctive concept to its parent conjunctive constituents. Defines a subsumption relationship as well as the definition of the conjunctive concept (i.e. the conjunction of all of its parent $\sqsubseteq_c$ links).
- $\sqsubseteq_d$ – A definitional link from a disjunctive concept to its child disjunctive constituents. Defines a subsumption relationship as well as the definition of the disjunctive concept (i.e. the disjunction of all of its child $\sqsubseteq_d$ links).

Figure 2 shows an example complete taxonomy involving these links.

It is also important here to mention a few issues regarding the treatment of primitive concepts in this algorithm implementation. Although it seems that we are not classifying primitives according to the intensional subsumptional definition, it turns out in fact that we are. First, it should be noted that primitive concepts can only be structurally subsumed by conjunctive concepts and they can only structurally subsume

Algorithm 1: $ClassifyKb(kb)$

```
begin
    // Add all primitive concepts and axioms
    Add ⊤, ⊤_rel, and ⊥ to the taxonomy ;
    Add all primitive kb concepts to the taxonomy ;
    foreach ((concept axiom a ⊑ b) ∈ kb) do
        Add the taxonomy link a ⊑_a b ;
    foreach (concept c ∈ taxonomy) do
        if (c has no axiomatic parent) then
            Add the taxonomy link c ⊑_s ⊤ ;
        if (c has no axiomatic child) then
            Add the taxonomy link ⊥ ⊑_s c ;
    // Add all relations and axioms
    Add all kb relations to the taxonomy ;
    foreach ((relation axiom r ⊑ s) ∈ kb) do
        Add the taxonomy link r ⊑_a s ;
    foreach ((relation r) ∈ taxonomy) do
        if (r has no axiomatic parent) then
            Add the taxonomy link r ⊑_s ⊤_rel;
    // Classify and add defined concepts
    foreach ((def d ≐ ...) ∈ kb) do
        Classify(d) ;
end
```

Algorithm 2: $Classify(target)$

```
begin
    // Recursively normalize and classify concept
    if (target is primitive) then return;
    Add target to the taxonomy ;
    NormalizeAndDefine(target) ;
    // Determine position in taxonomy
    mss ⟵ MSS(target) ;
    mgs ⟵ MGS(target) ;
    if (mss ∩ mgs = ∅) then
        // New structure so insert into taxonomy
        foreach (p ∈ mss) do
            Add the taxonomy link target ⊑_s p;
        foreach (c ∈ mgs) do
            Add the taxonomy link c ⊑_s target;
        foreach (p ∈ mss) do
            foreach (c ∈ mgs) do
                Remove the taxonomy link c ⊑_s p;
    else
        // target is equivalent to any a
        Choose any a ∈ mss ∩ mgs;
        Add the taxonomy link target ⊑_s a;
        Add the taxonomy link a ⊑_s target;
end
```

disjunctive concepts. Consequently when we classify defined concepts, the conjunctive mgs algorithm and disjunctive mss algorithm will respectively implement this definition by respectively finding any primitive children of conjunctive concepts and any primitive parents of disjunctive concepts. Thus, as we will see, this method of classifying all primitive concepts and then classifying all defined concepts will still yield a correct algorithm with respect to the intensional subsumption definition.

4.2 Classification of a concept

Given the above algorithm for classifying an entire knowledge base, we now define Algorithm 2 for classifying an individual structured concept. This algorithm is straightforward, first normalizing the concept, then finding its mss and mgs sets, and finally inserting it into the taxonomy and removing redundant structural links if necessary.[8] The only non-obvious component of classification is that the normalization algorithm must take care of recursively classifying constituents of the concept such as restrictions and the conjunctive components of the DNF definition. It is important to point out that unlike languages such as Classic [Borgida *et al.*, 1989], this mutual recursion between classification and normalization is required for correctness of concept normalization (this is due to the complex interactions than can occur between role restrictions, conjunction, and disjunction).

[8]Note that some of the redundant links could be axioms or definitions. For the representation we will use here, it is important to delete only structurally inferred links since axiomatic and definitional links carry important structural information in addition to their subsumption interpretation.

4.3 Normalization algorithm

Normalization is required to ensure that structural comparison of two defined concepts will yield all extensionally complete subsumption relationships. Here we discuss the individual steps involved in normalization and then proceed to specify the full algorithm.

1. *Expand definitions*: Any non-primitive concept must be replaced by its definition until only primitive concepts remain. For example, given the definition $(A \doteq B \sqcup C)$, the concept $D \sqcap A \sqcap \exists R.A$ can be expanded to $D \sqcap (B \sqcup C) \sqcap \exists R.A$. Note that the restriction referent A in $\exists R.A$ is not expanded *yet*; We will recursively apply this algorithm to all role restriction referents in a subsequent step.
2. *Convert to DNF*: We convert the concept to disjunctive normal form (i.e. a a disjunction of purely conjunctive elements) by distributing $\sqcup$ over $\sqcap$. For example, given the concept $(A \sqcup B) \sqcap C$ we distribute the $\sqcup$ to obtain $(A \sqcap C) \sqcup (B \sqcap C)$.
3. *Convert restrictions to canonical form*: For each role of the following type, perform the given conversion:
   a) $\exists R.C \longrightarrow \geqslant 1R.C$
   b) $\geqslant nR \longrightarrow \geqslant nR.\top$

c) $\leqslant nR \longrightarrow \leqslant nR.\top$

4. *Merge conjoined universal role restrictions*: If any $\forall$ role restrictions in the same conjunction are exactly the same except for their referent concepts, we merge them and conjoin their referent concepts. For example, we rewrite $\forall nR.C \sqcap \forall nR.D$ as $\forall nR.(C \sqcap D)$.
5. *Classify all role restrictions*: All role restriction referents should be classified.
6. *Rewrite tautologous and inconsistent role restrictions*: All role restrictions determined by classification to be equivalent to the following forms should be rewritten to their equivalent $\top$ or $\bot$ definitions:
   a) $\forall R.\top \longrightarrow \top$
   b) $\leqslant nR.\bot \longrightarrow \top$
   c) $\geqslant 0R.C \longrightarrow \top$
   d) $\geqslant nR.\bot$ where $n \geq 1 \longrightarrow \bot$
7. *Eliminate redundant conjoined elements*: Constituents that subsume other constituents within a conjunction should be removed. For example, if it is known that $A' \sqsubseteq A$, then the concept $(A \sqcap A') \sqcup B$ can be simplified to $A' \sqcup B$. As another example, if classification has inferred that $\geqslant n'R'.C'$ is subsumed by $\geqslant nR.C$ then the concept $\geqslant n'R'.C' \sqcap \geqslant nR.C$ will be simplified to $\geqslant n'R'.C'$.
8. *Merge, classify, rewrite, and eliminate disjoined universal role restrictions*: If two different disjoined sets of conjunctions are equivalent in every way except for a single universal role restriction in each which agree on their restriction type and relation, we disjoin the referent concepts in these restrictions and merge the two conjunctions.[9] Next we classify the newly disjoined restriction referent and rewrite it according to step 7 if it is equivalent to any of those forms. Next we check if the restriction (or its rewrite) is redundant with any other conjoined elements as in step 8 and eliminate it if it is redundant. Finally, we repeat these steps until no further disjunctive role merges are possible. Following is an example which makes use of many of these steps: Assume we are given $A' \sqsubseteq A$, $B' \sqsubseteq B$, and $R' \sqsubseteq R$, and the concept $(\forall R'.(A' \sqcup B') \sqcap \forall R.A) \sqcup (\forall R'.(A' \sqcup B') \sqcap \forall R.B)$. We first notice that the two components of the conjunction match except for the restrictions $\forall R.A$ and $\forall R.B$. Consequently we merge the two conjunctive components to form $\forall R'.(A' \sqcup B') \sqcap \forall R.(A \sqcup B)$. Next we classify the newly formed restriction referent $\forall R.(A \sqcup B)$ and find that it subsumes $\forall R'.(A' \sqcup B')$. This leads us to remove the redundant subsumer, leaving us with the final normalized concept $\forall R'.(A' \sqcup B')$ which is equivalent to the original concept.
9. *Build and classify conjunctive substructure*: Since a DNF definition is broken down into two levels, i.e. a disjunction of conjunctions, we build formal anonymous concepts for each of the conjunctions and recursively classify them. For example, the DNF concept $D \doteq (A \sqcap B) \sqcup (C \sqcap \exists R.(A \sqcup B))$ is replaced with the following definitions: $D \doteq Anon_1 \sqcup Anon_2$, $Anon_1 \doteq A \sqcap B$, $Anon_2 \doteq C \sqcap \exists R.Anon_3$, and $Anon_3 \doteq A \sqcup B$. Then for each anonymous conjunctive concept, we add $\sqsubseteq_c$ taxonomy links with the anonymous concept as child and its conjunctive constituents as parents. For example, given $Anon_1 \doteq A \sqcap B$, we add the two taxonomy links $Anon_1 \sqsubseteq_c A$ and $Anon_1 \sqsubseteq_c B$. Once each anonymous concept is defined, we then proceed to recursively classify it.
10. *Eliminate redundant disjoined elements*: Constituents that are subsumed by other constituents within a disjunction should be removed. For example, if it is known that $A' \sqsubseteq A$, then the concept $A \sqcup A' \sqcup B$ can be simplified to $A \sqcup B$. As another example, if classification has inferred that $\geqslant n'R'.C' \sqsubseteq geqslant nR.C$ then the concept $\geqslant n'R'.C' \sqcup \geqslant nR.C$ will be simplified to $\geqslant nR.C$.
11. *Build disjunctive concept structure*: Now that we have the conjunctive components of the DNF definition built and classified, we add $\sqsubseteq_d$ taxonomy links with the target concept as parent and its disjunctive anonymous constituents as children. From the above example, given $D \doteq Anon_1 \sqcap Anon_2$, we add the two taxonomy links $D \sqsubseteq_d Anon_1$ and $D \sqsubseteq_d Anon_2$. Once this is complete, the target concept is normalized and ready for classification.

Now that we have explained and provided examples for each of the individual steps, we present a formalized version of these steps in Algorithm 3.

4.4 Most specific subsumer (MSS)

As noted in the above explanation of taxonomic classification, only defined concepts (i.e. those with conjunctive or disjunctive definitions) and restrictions are passed to the classification algorithm. Consequently, in Algorithm 4, we define an mss algorithm to handle these respective concept structures.

First, however, we will provide a brief high-level description of the different components of this algorithm that effectively implement taxonomic inference of the intensional subsumption rules specified in Table 2. We give soundness and completeness results for these implementations in a subsequent section.

- *Conjunctively defined target*: If we are looking for the subsumers of a conjunctively defined concept, we can do this in two steps. In the first step we mark the target and all parents up to $\top$ with a distinct mark. Now, starting with a set containing the $\top$ concept, we simply search down the taxonomy from all elements in this set, adding any marked concept or any conjunctively defined concept that has all constituents marked. If during this search, any concept is a subsumer but has no children that are subsumers, this concept is a candidate mss.

[9] It was important to first remove redundant conjunctive constituents since this allows identical conjunctive constituents to be structurally identified.

Algorithm 3: $NormalizeAndDefine(target)$

begin

// Note: For an in-depth discussion of each of the
// following steps, please refer to section 4.3
if *($target$ is of the form $\leqslant nR.C$ or $\geqslant nR.C$)* **then**

// Just classify restriction referent (note: this
// restriction will be classified upon return)
$Classify(C)$;

else

// Expand concept structure, convert to DNF,
// and normalize restriction definitions
Expand non-primitive elements in $target$ def;
Convert $target$ def to DNF;
foreach *(role restriction $res \in target$)* **do**
  Convert res to canonical form;

// Merge, classify, rewrite and eliminate
// redundant conjoined elements
Merge all possible conjoined role restrictions;
foreach *(role restriction $res \in target$)* **do**
  $Classify(res)$;
  Rewrite res if tautologous or inconsistent ;
Eliminate redundant conjoined elements;

// Merge, classify, rewrite and eliminate
// redundant disjoined elements
while *(a disjoined role restriction can be merged)* **do**
  Merge the restriction to form res^*;
  $Classify(res^*)$;
  Rewrite res if tautologous or inconsistent;
  Eliminate res if redundant with conjoined elements;

// Define conjunctive substructure, classify,
// and remove redundant disjoined elements
Build anonymous concept defs for conjunctions;
foreach *(anonymous concept $anon$)* **do**
  Add $\sqsubseteq_c$ taxonomy links from $anon$ to to its conjunctive constituents;
  $Classify(anon)$;
  Eliminate $anon$ if redundant with other disjoined elements;

// Define disjunctive substructure (note - this
// concept will be classified upon return
Add $\sqsubseteq_d$ taxonomy links from the disjunctive anonymous concept constituents to $target$;

end

- *Disjunctively defined target*: If we are looking for the subsumers of a disjunctively defined concept, we must find other disjunctive concepts for which all constituents are subsumers of the target's constituents. To do this, we mark up from each of the target's constituents with a distinct mark and then collect all parent concepts of one of the constituents that contain all marks.[10] Additionally, we need to check for the special case of an mss conjunctive concept and we do this simply by checking to see if any conjunctively defined child of a disjunctive mss has all constituents within the mgs set. As a final note, it should be apparent that this algorithm also finds primitive parents of disjunctively defined concepts. Consequently, the disjunctive mss algorithm finds all primitive, conjunctive, and disjunctive subsumers of a disjunctive concept according to the intensional subsumption rules.
- *Restriction target*: There are a number of ways to find parent subsumers of restrictions and here we present a somewhat inefficient but relatively simple method for doing this. We assume that all concepts have been classified in an extensionally complete taxonomy so that finding a parent subsuming restriction according to the intensional subsumption rules simply consists of two steps. First, all restrictions making reference to the restriction referent or one of its parents is collected in a set. Then, all restrictions in this set are checked against the intensional subsumption rules to prune out the non-subsumers. (The final filtering step will ensure that only the most-specific relation subsumers are returned.)

As a final step during MSS, it is important to filter the mss candidate set for subsumers that are not most specific. It is generally difficult to order the subsumer search to guarantee that such anomalous mss's do not occur. Thus it is easiest as a final step to simply remove any superfluous concepts by removing any subsumers of other concepts within the set.

Note that for the structural comparisons in the intensional subsumption definition that seem to be ignored (i.e. *Primitive/Disjunctive*, *Conjunctive/Primitive*, and *Conjunctive/Disjunctive*), the intensional subsumption is inferrable directly through the transitive closure of axiom and definition links. Thus, it only takes primitive concept classification and definitional links to build a complete taxonomy for these cases.

Next we proceed to define the MSS algorithm in Algorithm 4. This algorithm makes reference to the following helper functions:

- $GetDirParents/Children(target, linkType)$: We use this function to get the directly linked parents/children of the specified link types for the target. Link type refers to one or more of $\sqsubseteq_a, \sqsubseteq_c, \sqsubseteq_d, \sqsubseteq_s$ and is abbreviated with a, b, c, d respectively. Returns the set of parents/children.
- $GetAllParents/Children(target, linkType)$: We use this function to get the recursive transitive closure of

[10] It should be apparent that any disjunctive mss concept must be a subsumer of every constituent of target, so it really does not matter which constituent we initiate our search from.

all specified link types starting from the target (searching upward for parents and downward for children). Returns the set of parents/children.

- $MarkAllParents/Children(target, linkType, markerID)$: We use this function to mark the recursive transitive closure of all specified link types starting from the target (searching upward for parents and downward for children). All concepts within this closure are given the specified marker id. Returns nothing.
- $ContainsMark(target, markerID)$: We use this function to determine whether the specified target has been marked with the specified marker id. Returns *true* or *false*.

4.5 Most general subsumee (MGS)

Just as in the above MSS algorithm, we define the MGS algorithm for conjunctively and disjunctively defined concepts and restrictions. One of the elegant symmetries in this algorithm is that it essentially mirrors the MSS algorithm in that it infers from the $\perp$ concept upward (rather than $\top$ downward) and the inference algorithms for conjunction and disjunction are effectively swapped. This similarity is no coincidence in fact and reflects the symmetry of conjunctive and disjunctive subsumption for the intensional definition. First, we will briefly discuss the details of inference for each type of concept structure:

- *Conjunctively defined target*: The algorithm for finding conjunctive mgs's is the same algorithm used to find disjunctive mss's except that now we pass a distinct mark down from each of the conjunctive constituents and collect all mgs's below one of the constituents. An additional subsumption check for disjunctive mgs's is included to fully implement the intensional subsumption rules. Also, as in disjunctive MSS, conjunctive MGS can find primitive concepts as children since this algorithm directly implements that rule from the intensional subsumption rules.
- *Disjunctively defined target*: This algorithm is essentially the same as conjunctive MSS except that we mark down from target to $\perp$, initiate our search at $\perp$, and expand our search frontier upward.
- *Restriction target*: This algorithm is essentially identical to the MSS algorithm for restrictions except that the direction of inference is reversed.

And finally, as in the MSS algorithm, we must also filter the candidate mgs set since there could be redundant elements within this set.

We now proceed to define the MGS algorithm in Algorithm 5 which refers to the same helper functions defined for the MSS algorithm.

4.6 Remarks on instance classification

We have omitted a discussion of instance classification but this would simply require a slightly modified subset of the algorithm given here.

The goal of instance classification is to find the most specific concept subsumers for a given instance. An instance

Algorithm 4: $MSS(target)$

begin
  // Find the subsumers of target
  $mss \longleftarrow \emptyset$;
  switch *(type of* $target$*)*
    case *(* $target$ *is conjunctively defined)*
      $MarkAllParents(target, a|c|d|s, 1)$;
      $frontier \longleftarrow \{\top\}$;
      while $(frontier \neq \emptyset)$ **do**
        Remove $c \in frontier$;
        $dc \longleftarrow GetDirChildren(c, a|c|d|s)$;
        foreach $(cld \in dc)$ **do**
          if *(*cld *is primitive and marked with 1 or* cld *is conjunctive and all* $GetDirParents(cld, c)$ *are marked with 1)* **then**
            $frontier \longleftarrow frontier \cup cld$;
        if *(no child subsumee exists)* **then**
          $mss \longleftarrow mss \cup cld$;
    case *(* $target$ *is disjunctively defined)*
      $cons \longleftarrow GetDirChildren(target, d)$;
      for $(i = 1 \ldots |cons|)$ **do**
        $MarkAllParents(cons_i, a|c|d|s, i)$;
      $frontier \longleftarrow \{cons_1\}$;
      while $(frontier \neq \emptyset)$ **do**
        Remove $c_1 \in frontier$;
        if *(*c_1 *contains all* $|cons|$ *marks)* **then**
          $mss \longleftarrow mss \cup c_1$;
          $dc \longleftarrow GetDirChildren(c_1, c)$;
          foreach $(c_2 \in dc)$ **do**
            if *(all* $GetDirParents(c_2, c)$ *contain all* $|cons|$ *marks)* **then**
              $mss \longleftarrow mss \cup c_2 \setminus c_1$;
        else
          $frontier \longleftarrow frontier \cup GetDirParents(c_1, a|c|d|s)$;
    case *(* $target$ *is a restriction)*
      $c_1 \longleftarrow$ { concept referent of $target$ } ;
      $par \longleftarrow c_1 \cup GetAllParents(c_1, a|c|d|s)$;
      foreach $(c_2 \in par)$ **do**
        foreach *(restriction* r *with concept referent* c_2*)* **do**
          if *(*$r \neq target$ *and* r *subsumes* $target$ *according to intensional subsumption rules)* **then**
            $mss \longleftarrow mss \cup r$;
  // Filter out non-most specific subsumers
  foreach $(c \in mss)$ **do**
    $mss \longleftarrow mss \setminus GetAllParents(c, a|c|d|s)$;
  return mss;
end

Algorithm 5: $MGS(target)$

```
begin
    // Find the subsumers of target
    mgs ⟵ ∅;
    switch (type of target)
        case ( target is conjunctively defined )
            cons ⟵ GetDirParents(target, c);
            for (i = 1 ... |cons|) do
                MarkAllChildren(cons_i, a|c|d|s, i);
            frontier ⟵ {cons_1};
            while (frontier ≠ ∅) do
                Remove c_1 ∈ frontier;
                if (c_1 contains all |cons| marks) then
                    mgs ⟵ mgs ∪ c_1;
                    dp ⟵ GetDirParents(c_1, d);
                    foreach (c_2 ∈ dp) do
                        if (all GetDirChildren(c_2, d)
                        contain all |cons| marks) then
                            mgs ⟵ mgs ∪ c_2 \ c_1;
                else
                    frontier ⟵ frontier ∪
                    GetDirChildren(c_1, a|c|d|s);
        case ( target is disjunctively defined )
            MarkAllChildren(target, a|c|d|s, 1);
            frontier ⟵ {⊥};
            while (frontier ≠ ∅) do
                Remove c ∈ frontier;
                dc ⟵ GetDirParents(c, a|c|d|s);
                foreach (par ∈ dc) do
                    if (par is primitive and marked
                    with 1 or par is disjunctive and
                    all GetDirChildren(cld, d) are
                    marked with 1) then
                        frontier ⟵ frontier ∪ par;
                if (no parent subsumer exists) then
                    mgs ⟵ mgs ∪ par;
        case ( target is a restriction )
            c_1 ⟵ { concept referent of target };
            cld ⟵ c_1 ∪ GetAllChildren(c_1, a|c|d|s);
            foreach (c_2 ∈ cld) do
                foreach (restriction r with concept re-
                ferent c_2) do
                    if (r ≠ target and r is subsumed
                    by target according to intensional
                    subsumption rules) then
                        mgs ⟵ mgs ∪ r;
    // Filter out non-most specific subsumees
    foreach (c ∈ mgs) do
        mgs ⟵ mgs \ GetAllChildren(c, a|c|d|s);
    return mgs;
end
```

structure will simply be a primary instance with *type-of* links to various concepts. In addition, this primary instance will have relation links to other instances of the same general structure. To classify such an instance, one need only run the conjunctive portion of MSS classification using the instance concept types as the constituents instead of the instances themselves. This should yield a structural algorithm for instance classification with the same soundness and completeness guarantees as that of the language being classified.

4.7 Remarks on incremental classification

For reasons of simplicity of presentation and analysis, we have chosen to provide a classification algorithm that requires full knowledge of all kb contents prior to the start of classification. This is a somewhat unreasonable assumption for the real world since we would like to add knowledge incrementally as it is encountered. It would not be difficult to add such enhancements - one would simply need to add classification rules for primitive concepts since their conjunctive subsumers or disjunctive subsumees may have already been classified. Otherwise, conversion of this algorithm to an extensionally sound and complete incremental version for $\mathcal{L}_1$ or a slightly incomplete incremental algorithm for $\mathcal{L}_2$ would be relatively straightforward.

5 Algorithm properties

In the following sections, we prove various properties of this algorithm regarding soundness, completeness, and time complexity.

5.1 Soundness and completeness

Soundness for classification in $\mathcal{L}_1$ and $\mathcal{L}_2$

Before we prove soundness of classification, it is first important to establish three important theorems:

5.1 Normalization preserves the extension of a concept if classification is sound for its constituents.

5.2 The classification algorithm correctly implements the intensional subsumption definition for normalized concepts.

5.3 The intensional subsumption definition is extensionally sound.

From these theorems, it is relatively simple to prove the following two additional lemmas:

5.4 The classification algorithm is extensionally sound for normalized concepts.

5.5 All concepts are extensionally equivalent to their normalized counterparts.

And from these lemmas, it is relatively straightforward to show that the overall classification algorithm is sound.

Theorem 5.1 *Normalization preserves the extension of a concept (i.e. if concept A has normalized form A' then $A \equiv A'$) if classification is sound for its constituents.*

To prove this, we show that each normalization step preserves extensional equivalence to the concept before the step was executed:

1. *Expand definitions*: Since we are replacing concepts by their equivalent definitions, extensional equivalence is preserved.
2. *Convert to DNF*: Since the DNF format is known to be logically equivalent to the original format, extensional equivalence is preserved.
3. *Convert restrictions to canonical form*: Based on the previously stated role conversions, it is trivial to show that every instance which satisfies the role restriction on the left will satisfy that on the right and vice versa.
4. *Merge conjoined universal role restrictions*: It is straightforward to show that any instance which satisfies $\forall R.A \sqcap \forall R.B$ must also satisfy $\forall R.(A \sqcap B)$ and vice versa. Clearly an instance of the former concept must restrict all objects related via R to be an instance of both A and B and this is exactly what the second concept states so equivalence is maintained.
5. *Classify all role restrictions*: This step does not affect concept structure and therefore trivially maintains equivalence.
6. *Rewrite tautologous and inconsistent role restrictions*: Obviously these role restrictions have the same extension as the $\top$ and $\bot$ concepts that replace them.
7. *Eliminate redundant conjoined elements*: If one constituent is found to be subsumed by another conjoined constituent via a sound classification procedure, the extension of that conjunction is at most the extension of that of the subsumee. Since the extension of the subsumer is a superset of the subsumee and cannot further constrain the extension of the conjunction, the subsumer can be safely eliminated from the conjunction without changing the extension of the concept.
8. *Merge, classify, rewrite, and eliminate disjoined universal role restrictions*: It is straightforward to show that any instance that satisfies $(C \sqcap \forall R.A) \sqcup (C \sqcap \forall R.B)$ must also satisfy $C \sqcap \forall R.(A \sqcup B)$ and vice versa. Clearly, an instance of the former concept must be an instance of both C and restrict any relations via R to be an instance of A or B depending on which part of the disjunction is classified. This is clearly what the second concept states so equivalence is maintained. To show that equivalence is maintained for the classification, rewriting, and elimination steps we refer to steps 6–7.
9. *Build and classify conjunctive substructure*: This step is simply for defining the concept within the taxonomy and has no effect whatsoever on concept structure.
10. *Eliminate redundant disjoined elements*: If one constituent is found to subsume another disjoined constituent via a sound classification procedure, the extension of that disjunction is at least the extension of that of the subsumer. Since the extension of the subsumee is a subset of the subsumer and cannot further contribute to the extension of the disjunction, the subsumee can be safely eliminated from the disjunction without changing the extension of the concept.
11. *Build disjunctive concept structure*: Again, this step is simply for defining the concept within the taxonomy and has no effect on concept structure.

Since each step preserves extensional equivalence of the concept if classification is sound, the overall normalization algorithm must preserve extensional equivalence if classification is sound for its constituents. □

Theorem 5.2 *The classification algorithm correctly implements the provided intensional subsumption definition for normalized concepts.*

A formal proof of correctness would be quite involved so we will instead provide a proof sketch that should make the formalization relatively obvious.

We prove correctness of classification including normalization with respect to the intensional subsumption definition by showing that any subsumption returned by the classification algorithm[11] satisfies the intensional subsumption definition and vice versa. Both proofs rely on structural induction on the complexity of the concept being classified and make the recursive case assumption that correctness holds for any concepts of lesser complexity.

To make the proof easier, we assume that the concepts being compared via the classification algorithm and the intensional subsumption definition have both been normalized – this simplifies the correctness proof since concepts on both sides can be assumed to have the same structure. Furthermore this is a valid strategy since by structural induction, if the algorithms agree on subsumption for all concepts less complex than the ones being classified, they will also agree on the structure of the normalized concept.[12] This results from the fact that normalization relies only on the subsumption relationships between constituents of the concept being classified (i.e. lower complexity concepts than the one being classified) and by structural induction, we can assume that the classification algorithm and intensional subsumption definition agree on these subsumptions.

In the following proof, we start first with the base cases and proceed to the recursive cases showing that both directions of the equivalence *classification* $\leftrightarrow$ *intensional subsumption def* hold for each case. Furthermore, whenever two concepts are listed separated by a */*, the first concept denotes the subsumee in question which we label A and the second denotes the subsumer in question which we label B.

- *(Base) Role restriction combination not mentioned in Table 2*:

  $\leftrightarrow$: Neither the algorithm or the intensional subsumption definition make any subsumption inferences for role restrictions not explicitly mentioned in Table 2.

- *(Base) Primitive/Primitive*:

[11] Here a subsumption returned by the classifiation algorithm is any subsumption that can be inferred from the transitive closure of subsumption links in the taxonomy.

[12] Note that the base cases do not require normalization, i.e. see the first line of Algorithm 3. This is quite crucial to the validity of the proof strategy given here.

$\leftrightarrow$: $ClassifyKb()$ handles this inference and clearly any concept A found to be subsumed via transitive role closure of $\sqsubseteq_a$ and $\sqsubseteq_s$ links inserted by this procedure must satisfy the intensional definition. Likewise, the intensional definition is effectively implemented by $ClassifyKb()$.

- *(Recursive) Primitive/Conjunctive*:

  $\leftrightarrow$: When the conjunctive portion of the MGS algorithm runs for concept B, it will clearly find any concept A that is subsumed according to the intensional subsumption definition. Thus, the conjunctive MGS algorithm and this intensional subsumption rule clearly compute the same thing.

- *(Recursive) Primitive/Disjunctive*:

  $\leftrightarrow$: This case reduces to a subsumption test between the primitive concept and the primitive constituents of the disjunctive concept that is effectively handled by the *Primitive/Primitive* case. Correspondence for the *Primitive/Primitive* case has already been shown, therefore the classification algorithm and intensional subsumption definition compute the same subsumptions for this case.

- *(Recursive) Conjunctive/Primitive*:

  $\leftrightarrow$: This case reduces to a subsumption test between the primitive concept and the primitive constituents of the disjunctive concept that is effectively handled by the *Primitive/Primitive* case. Correspondence for the *Primitive/Primitive* case has already been shown, therefore the classification algorithm and intensional subsumption definition compute the same subsumptions for this case.

- *(Recursive) Conjunctive/Conjunctive*:

  Since this structural comparison along with the *Disjunctive/Disjunctive* comparison likely accounts for the majority of subsumptions between structured concepts, we examine this case a little more in-depth to show inference equivalence of the intensional subsumption definition and the classification algorithm.

  $\rightarrow$: This type of subsumption inference can be made in two places in the classification algorithm, i.e. the conjunctive portions of the MSS and MGS algorithms. For MSS, if a conjunctive concept is inferred to be a parent then its constituents must all be subsumers of the consituents of the subsumee. This is exactly what the intensional rule states for subsumption between conjunctive concepts. If MGS infers a conjunctive concept as child, the child's constituents must contain subsumees of all the parent concept's constituents. Again, this is exactly what the intensional rule states. Consequently, if a concept is inferred to be a conjunctive subsumer of subsumee via classification, it must satisfy the intensional subsumption definition.

  $\leftarrow$: To show that no intensional rule inferences are missed by the conjunctive portions of either the MSS or MGS algorithms, it suffices to show that both algorithms examine all possible candidates. The MSS algorithm starts at $\top$ and examines all concepts which satisfy the intensional rule (terminating only in places where a concept is not a subsumer and therefore neither are its children), consequently, no concept could be missed. The MGS algorithm examines all concepts that are children of one of the target's constituents. Since a subsumee of this consituent is required to be a constituent of all MGS subsumees, the MGS algorithm also examines all possible MGS candidates. Given that the MGS candidate checking algorithm exactly implements the intensional semantics, no candidate could be missed by MGS. Thus, any subsumption between conjunctively defined concepts satisfying the intensional subsumption definition can be inferred through the classification algorithm.

- *(Recursive) Conjunctive/Disjunctive*:

  $\leftrightarrow$: This case reduces to a subsumption test between the primitive concept and the primitive constituents of the disjunctive concept that is effectively handled by the *Primitive/Primitive* case. Correspondence for the *Primitive/Primitive* case has already been shown, therefore the classification algorithm and intensional subsumption definition compute the same subsumptions for this case.

- *(Recursive) Disjunctive/Primitive*:

  $\leftrightarrow$: When the disjunctive portion of the MSS algorithm runs for concept A, it will clearly find any concept B that is a subsumer according to the intensional subsumption definition. Thus, the disjunctive MSS algorithm and this intensional subsumption rule clearly compute the same subsumptions.

- *(Recursive) Disjunctive/Conjunctive*:

  $\leftrightarrow$: This intensional subsumption rule is determined in two places in the classification algorithm. If concept B is classified after concept A, then A will be found as a subsumee of B in the conjunctive portion of the MGS algorithm (there is a special test in this algorithm for this subsumption). If concept A is classified after concept B, then B will be found as a subsumer of A in the disjunctive portion of the MSS algorithm (again, there is a special test in this algorithm for this subsumption). A careful analysis of both of these algorithms should reveal that they both correctly implement the intensional subsumption rule for this concept structure combination. That is, they examine all possible candidates and correctly retain those that satisfy the definition. Thus, this portion of the intensional subsumption definition and the classification algorithm compute the same subsumptions.

- *(Recursive) Disjunctive/Disjunctive*:

  $\leftrightarrow$: Since the classification algorithm and intensional definitions are completely symmetric with the *Con-*

junctive/Conjunctive case, we can use the exact same proof structure for that case to prove equivalence of this case. Thus, we will omit further formal proof of this case, instead referring to the *Conjunctive/Conjunctive* proof with the direction of inference reversed.

- *(Recursive) Role restriction/Role restriction*:

  $\leftrightarrow$: This is the one part of the algorithm that was not optimized for sake of brevity and thus it is relatively straightforward to prove equivalence of the intensional subsumption definition and the algorithm in this case. Clearly, the restriction portion of the MSS and MGS algorithms examine all possible candidates that might conform to the intensional subsumption definition. This is because the examine *any* role restriction that concept referents which are subsumers or subsumees of the target concept being classified. Since this is a requirement for role restriction subsumption (in either case), we can be sure that all possible candidates are examined. From this point, all candidates are tested directly against the intensional subsumption definition. Thus the classification algorithm and intensional subsumption definition infer the same subsumptions.

Consequently, by structural induction we have shown that the classification algorithm given in Algorithms 1–5 and the intensional subsumption definition given in Table 2 are equivalent in terms of subsumption inference. Therefore, we can deduce that the classification algorithm correctly implements the intensional subsumption definition for $\mathcal{L}_1$ and $\mathcal{L}_2$. □

Theorem 5.3 *The provided intensional subsumption definition is extensionally sound.*

For every structural comparison rule given in Table 2, it is straightforward to show that any instance satisfying concept B must also satisfy concept A. In the following listing, we provide the type of concept A followed by the type of concept B. Additionally, if A or B are conjunctively or disjunctively defined, we refer to their constituents as A_i or B_j respectively.

- *Primitive/Primitive*: Clearly by the constraint axioms linking concept A and concept B, every instance in $A^{\mathcal{I}}$ must also be in $B^{\mathcal{I}}$.
- *Primitive/Conjunctive*: If A is subsumed by every B_j then clearly every instance in $A^{\mathcal{I}}$ is in every $B_j^{\mathcal{I}}$ and therefore in the intersection of all $B^{\mathcal{I}}$.
- *Primitive/Disjunctive*: If A is subsumed by some B_j then clearly the following relationship holds by definition and subsumption constraints: $A^{\mathcal{I}} \subseteq B_j^{\mathcal{I}} \subseteq B^{\mathcal{I}}$.
- *Conjunctive/Primitive*: If some A_i is subsumed by B then clearly the following relationship holds by definition and subsumption constraints: $A^{\mathcal{I}} \subseteq A_i^{\mathcal{I}} \subseteq B^{\mathcal{I}}$.
- *Conjunctive/Conjunctive*: This rule requires that every B_j subsumes some A_i. Additionally, from the definition, any instance in the intersection of $A_i^{\mathcal{I}}$, i.e. $A^{\mathcal{I}}$, must obviously be in every $A_i^{\mathcal{I}}$. Since an instance in every $A_i^{\mathcal{I}}$ must necessarily be in each $B_j^{\mathcal{I}}$ due to the subsumption constraint, this instance must also be in the intersection $B^{\mathcal{I}}$. Thus, any instance in $A^{\mathcal{I}}$ must necessarily be in $B^{\mathcal{I}}$.
- *Conjunctive/Disjunctive*: This rule requires that some B_j subsumes some A_i. Since B subsumes B_j and A_i subsumes A by default, clearly by transitivity of subsumption, the following relation holds: $A^{\mathcal{I}} \subseteq A_i^{\mathcal{I}} \subseteq B_j^{\mathcal{I}} \subseteq B^{\mathcal{I}}$. Thus, any instance of A must also be an instance of B.
- *Disjunctive/Primitive*: If B subsumes every A_i, then clearly every instance in the union of all $A_i^{\mathcal{I}}$, (i.e. $A^{\mathcal{I}}$), is also necessarily in $B^{\mathcal{I}}$.
- *Disjunctive/Conjunctive*: This rule requires that every B_j must subsume every A_i. It should also be clear that any instance in the disjunction $A^{\mathcal{I}}$ must be in one of the $A_i^{\mathcal{I}}$. Since every $A_i^{\mathcal{I}}$ must be a subset of every $B_j^{\mathcal{I}}$, clearly any instance in $A_i^{\mathcal{I}}$ is also in every $B_j^{\mathcal{I}}$ and therefore the intersection $B^{\mathcal{I}}$. Thus, any instance in $A^{\mathcal{I}}$ must necessarily be in $B^{\mathcal{I}}$.
- *Disjunctive/Disjunctive*: This rule requires that every A_i must be subsumed by some B_j. Additionally, from the definition, an instance in $A^{\mathcal{I}}$ must be in some $A_i^{\mathcal{I}}$. Since an instance in $A_i^{\mathcal{I}}$ must be in $B_j^{\mathcal{I}}$ by the subsumption constraint, it must also be in the disjunction $B^{\mathcal{I}}$. Thus, it should be clear that any instance in $A^{\mathcal{I}}$ must necessarily be in $B^{\mathcal{I}}$.
- *Role restriction/Role restriction*: Given the rules for role restriction subsumption, it is straightforward to show that each part of the rule (i.e. relation subsumption, restriction referent subsumption, and number comparison if relevant) restrict the set of instances in $A^{\mathcal{I}}$ to be a necessary subset of those in $B^{\mathcal{I}}$.
- *Concept/Role restriction* or Role restriction/Concept: There is no such rule for this combination. Consequently, *false* is returned by default and this is trivially sound.
- *Role restriction combination not mentioned in Table 2*: This rule only returns *false* which is trivially sound.

Consequently, each individual intensional subsumption rule is sound, and false is returned otherwise, thus the intensional subsumption definition is sound. □

Lemma 5.4 *The classification algorithm is extensionally sound for normalized concepts in $\mathcal{L}_1$ and $\mathcal{L}_2$.*

We know from theorem 5.2 that the classification algorithm correctly implements the intensional semantics, assuming both are using normalized concepts A' and B' derived from A and B. Thus, if classification infers $A' \sqsubseteq B'$ then by the definition in Table 2 $A' \sqsubseteq_i B'$. Second, we know from theorem 5.3 that the intensional subsumption definition is extensionally sound. Thus, we know that $A'^{\mathcal{I}} \subseteq B'^{\mathcal{I}}$. Thus, the classification algorithm is extensionally sound for normalized concepts. □

Lemma 5.5 *The normalized version of a concept is extensionally equivalent to its unnormalized version for all concept structures in $\mathcal{L}_1$ and $\mathcal{L}_2$.*

To prove equivalence between all normalized and unnormalized concepts using the classification algorithm, we show this by structural induction on the complexity of a concept:

- *Base case*: We start with a base case of primitive concepts for which this trivially holds. Since classification of concepts is sound for all normalized concepts, it must also be sound for unnormalized primitive concepts since they are syntactically identical to their normalized counterparts.
- *Recursive case*: We assume that here we are trying to classify some target concept for which we can assume its constituents are all equivalent to their normalized counterparts via the inductive hypothesis. Since equivalence holds for the constituents, we know that extensionally sound classification as given in lemma 5.4 for normalized constituents yields extensionally sound classification for the unnormalized constituents. Thus via theorem 5.1, we know that equivalence must be maintained for the target concept.

Thus, equivalence holds for all normalized and unnormalized concepts using the provided classification algorithm. □

Theorem 5.6 *Subsumption inference for concepts in $\mathcal{L}_1$ and $\mathcal{L}_2$ using the provided classification algorithm is extensionally sound.*

Given lemmas 5.4 and 5.5, it is a fairly straightforward argument to show that the classification algorithm is extensionally sound. We simply need to show that if classification infers $A \sqsubseteq B$ then it must hold that $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$.

Lemma 5.4 implies that if classification infers $A \sqsubseteq B$, then $A'^{\mathcal{I}} \subseteq B'^{\mathcal{I}}$ must hold for the normalized versions of the concepts. And from lemma 5.5, it is straightforward to show that we can infer $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ from $A'^{\mathcal{I}} \subseteq B'^{\mathcal{I}}$. Thus, classification is extensionally sound for $\mathcal{L}_1$ and $\mathcal{L}_2$. □

Completeness for classification in $\mathcal{L}_1$

To show completeness of subsumption inference in $\mathcal{L}_1$, we need to show one main property, i.e. that the intensional subsumption definition given in Table 2 is complete for normalized concepts. From this, it is relatively straightforward from previously proved theorems and lemmas to establish that classification is complete for all concepts in $\mathcal{L}_1$.

Theorem 5.7 *Subsumption inference for normalized concepts in $\mathcal{L}_1$ using the intensional subsumption definition given in Table 2 is extensionally complete.*

Here we rely on the fact that we know that a concept is normalized and proceed to show via structural induction that if any subsumption inference is not made according to the intensional subsumption definition, then there always exists a countermodel for that subsumption inference.

In the following listing, we provide the type of concept A followed by the type of concept B. Additionally, if A or B is conjunctively or disjunctively defined, we refer to its constituents as A_i or B_j respectively.

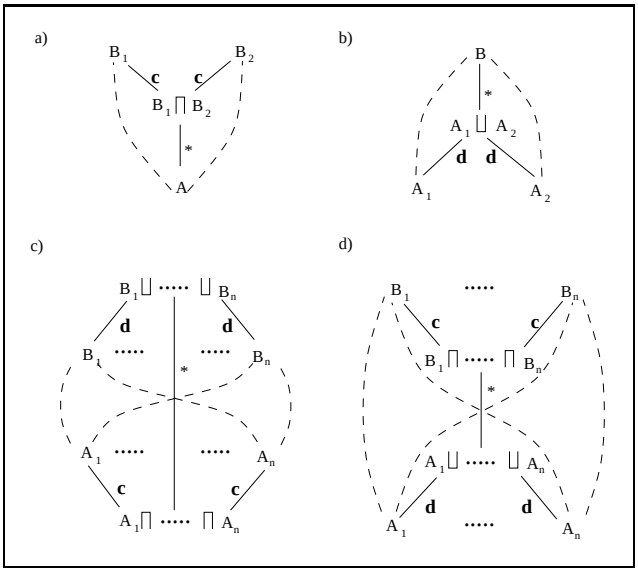

Figure 3: Sample diagrams to illustrate parts of the completeness proof for the following *subsumee(A)/subsumer(B)* concept structure pairs: a) *Primitive/Conjunctive* b) *Disjunctive/Primitive* c) *Conjunctive/Disjunctive* d) *Disjunctive/Conjunctive*. Solid lines indicate the subsumptions due to definitions, solid lines with a star indicate the intended structural subsumption inference, and dashed lines indicate the subsumptions of which all or a subset must hold true in order for the intended structural subsumption relation to hold. To avoid an excess of arrows, it is assumed that a vertically higher concept subsumes a vertically lower concept if a link exists.

- *Primitive/Primitive*: Clearly, if a subsumption inference was not made then there was no chain of axiomatic or structural subsumption links leading from A up to B. Consequently, if no such chain exists then it is easy to construct a model for $A^{\mathcal{I}}$ that contains an instance not in $B^{\mathcal{I}}$ but is still consistent with the knowledge base constraints.
- *Primitive/Conjunctive*: If A is not subsumed by every B_j then clearly not every instance in $A^{\mathcal{I}}$ is in every $B_j^{\mathcal{I}}$. Thus, not every instance in $A^{\mathcal{I}}$ is in the intersection $B^{\mathcal{I}}$ and this demonstrates a way to construct a countermodel. Figure 3a helps one to visualize the constraints for this proof.
- *Primitive/Disjunctive*: This rule requires that some B_j subsume A. If this is not the case, then a countermodel can be constructed by the following procedure: Simply take a valid model and add a new instance i_1 to A but not to any B_j. This instance clearly satisfies $i_1 \in A^{\mathcal{I}}$. However, since no B_j subsume A, $i_1 \notin \bigcup B_j^{\mathcal{I}}$ can hold and by definition $i_1 \notin B^{\mathcal{I}}$ can hold as well. This clearly violates $A \sqsubseteq B$ and thus a countermodel exists for any subsumption failing to meet this rule. Figure 3b helps one to visualize the constraints for this proof.
- *Conjunctive/Primitive*: This rule requires that some A_i

be subsumed by B. If this is not the case, then a countermodel can be constructed by the following procedure: Simply take a valid model and add a new instance i_1 to each A_i but not to B. This instance satisfies $i_1 \in \bigcap A_i^{\mathcal{I}}$ and by definition, $i_1 \in A^{\mathcal{I}}$. However, since no A_i are subsumed by B, $i_1 \notin B^{\mathcal{I}}$ can hold. This clearly violates $A \sqsubseteq B$ and thus a countermodel exists for any subsumption failing to meet this rule. Figure 3b helps one to visualize the constraints for this proof.

- *Conjunctive/Conjunctive*: This rule requires that every B_j subsumes some A_i. If this rule does not hold then it is straightforward to build a countermodel. We simply assume that A is non-empty and for the B_j which does not subsume some A_i, we set $B_j^{\mathcal{I}} \equiv \emptyset$. Then clearly $B^{\mathcal{I}} \equiv \bigcap B_j^{\mathcal{I}} \equiv \emptyset$ while $A^{\mathcal{I}}$ is not empty and thus B does not subsume A in this model.
- *Conjunctive/Disjunctive*: This rule requires that some B_j subsumes some A_i. If this does not hold then clearly it is possible to construct a countermodel for this subsumption. Figure 3c shows in part why this is the case: If no A_i is subsumed by a B_j then it is easy to construct a model with an instance i_1 in all $A_i^{\mathcal{I}}$ but no $B_j^{\mathcal{I}}$ such that $i_1 \in \bigcap A_i^{\mathcal{I}}$ but $i_1 \notin \bigcup B_j^{\mathcal{I}}$. (This is not perhaps as straightforward as it sounds. See the note below on why this *must* hold for all normalized concepts.) Thus, by the definition of A and B, a countermodel clearly exists where an instance in $A^{\mathcal{I}}$ is not in $B^{\mathcal{I}}$ and the subsumption cannot hold.

★ There is one extremely important note to be made here: This is the one case of structural subsumption which requires that the concepts be in disjunctive normal form. Otherwise, if both concepts had structured components, they may not obey this rule but still indirectly constrain each other to be subsumers. An example of this is given in Figure 4. In this case, $A \sqcup (B \sqcap C)$ is not inferred to be a parent of $(A \sqcup C) \sqcap (A \sqcap B)$ and yet these concepts are obviously equivalent. So, why does this occur? Because any instance in $(A \sqcup C) \sqcap (A \sqcap B)$ must belong to both $(A \sqcup C)$ and $(A \sqcup B)$. Consequently, if the instance belongs to either A in these disjunctions, $A \sqcup (B \sqcap C)$ is satisfied. And if the instance belongs to neither A in the disjunctions then it must belong to both $B \sqcap C$ thus also satisfying $A \sqcup (B \sqcap C)$.

Thus, extensionally there clearly is a subsumption relationship. However, with a little thought it becomes apparent that if either conjunctions or disjunctions are restricted to have primitive constituents as in DNF or CNF, then such a structurally unidentifiable subsumption is impossible (i.e. adding an instance to all primitive constituents of a concept could not indirectly add the instance to any other concepts without adding it directly to its constituents or a constituent of one of its constituents).[13] Consequently, this underscores the need for DNF normalization to prove completeness here.[14]

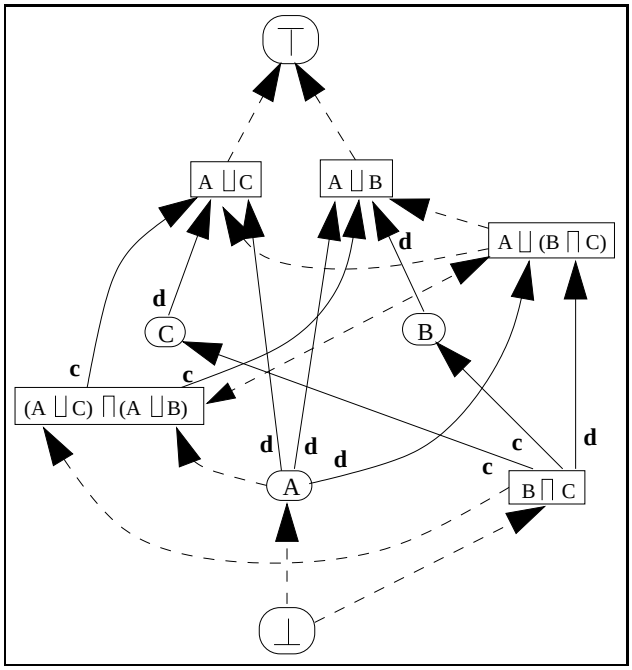

Figure 4: Example case where lack of DNF normalization leads to a structurally unidentifiable subsumption. Dotted lines indicate extensionally inferred subsumptions, solid lines indicate definitionally inferred subsumptions. See the section denoted with ★ for a discussion of this example.

- *Disjunctive/Primitive*: If B does not subsume every A_i, then clearly every instance in the union of all $A_i^{\mathcal{I}}$, (i.e. $A^{\mathcal{I}}$), is not necessarily in $B^{\mathcal{I}}$. Thus it is easy to construct a countermodel that demonstrates this. Figure 3b helps one to visualize the constraints for this proof.
- *Disjunctive/Conjunctive*: This rule requires that every B_j must subsume every A_i. If this rule does not hold then it is possible to construct a countermodel for this subsumption. We demonstrate this by referring to Figure 3d: Clearly if there is some A_i in this diagram which does not have a subsumption link to all B_j then it is easy to construct a model with an instance i_1 such that $i_1 \in A_i^{\mathcal{I}}$ but $i_1 \notin B_j^{\mathcal{I}}$. Then clearly $i_1 \in \bigcup A_i^{\mathcal{I}}$ but $i_1 \notin \bigcap B_j^{\mathcal{I}}$ and by the definition of A and B, a countermodel clearly exists where an instance in $A^{\mathcal{I}}$ is not in $B^{\mathcal{I}}$ and the subsumption cannot hold.
- *Disjunctive/Disjunctive*: This rule requires that every A_i must be subsumed by some B_j. If this rule does not hold then it is straightforward to build a countermodel. We do this with the following procedure: First, take a valid model of $A \sqsubseteq B$. Next, add some new instance i_1 not mentioned anywhere in the current model and add it to $A_i^{\mathcal{I}}$ for the A_i which breaks the subsumption rule above. Now, clearly $i_1 \in \bigcup A_i^{\mathcal{I}}$ but $i_1 \notin \bigcup B_j^{\mathcal{I}}$. Thus, by the definition of A and B, a countermodel clearly

[13] This is a bit tricky so it is worth taking the time to understand.

[14] CNF normalization could have been chosen as well but this would conflict with the upcoming discussion of the expected DNF structure of concepts on the Semantic Web.

exists where an instance in $A^{\mathcal{I}}$ is not in $B^{\mathcal{I}}$ and the subsumption cannot hold.

- *Role restriction/Role restriction*: Given the rules for role restriction subsumption, it is straightforward to show that each part of the rule (i.e. relation subsumption, restriction referent subsumption, and number comparison if relevant) restrict the set of instances in $A^{\mathcal{I}}$ to be a necessary subset of those in $B^{\mathcal{I}}$. If a subsumption inference is not made then clearly one of these subrules does not hold and it is easy to construct a countermodel which disproves the subsumption.

  Here we briefly discuss some issue involving normalization and its completeness implications for subsumption of role restrictions:

  - Note that while role restriction merging is required to achieve as much completeness as possible in $\mathcal{L}_2$, it is only used for the $\forall R.C$ restriction which is not in $\mathcal{L}_1$.
  - Role conversion is only important for inference of $\geqslant 0R.C \longrightarrow \top$ and completeness implications for this normalization are discussed below in the *Concept/Role restriction* section.
  - Redundant role removal is only important for correct role restriction merging which does not apply for $\mathcal{L}_1$ and for inference of pure $\top$ and $\bot$ concepts which is only important for role conversion in $\mathcal{L}_2$. In general, the presence of redundant role restrictions in $\mathcal{L}_1$ does not affect completeness and this can be easily seen by examining a few example cases.

  Thus we will not discuss the effect of role restriction normalization on completeness since it is not important for this proof.

- *Role restriction combination not mentioned in Table 2*: Note that we only provide subsumption rules for role restrictions of the same type. Furthermore, we note that in $\mathcal{L}_1$ (i.e. the language we are proving completeness for here), all of the role restrictions $\exists R.C$, $\geqslant nR$, and $\geqslant nR.C$ can be normalized to $\geqslant nR.C$. Thus, after normalization for $\mathcal{L}_1$, this rule becomes vacuous and is never used.[15] Consequently, this rule has no effect on completeness for $\mathcal{L}_1$.

- *Role restriction/Concept* or *Concept/Role restriction*: A quick look at the extensional semantics for role restrictions and concepts should make it clear that it would be quite easy to construct a countermodel for any subsumption of this type, except for the case where a role restriction happens to be equivalent to $\top$ or $\bot$. However, all of these cases have been accounted for in normalization (actually only one case is applicable for $\mathcal{L}_1$, i.e. $\geqslant 0R.C \longrightarrow \top$). Consequently, any other subsumptions between structures of this type would clearly have a countermodel.

[15] We did prove soundness of this rule in theorem 5.3 but there we were looking at both $\mathcal{L}_1$ and $\mathcal{L}_2$

Since a countermodel always exists if the intensional subsumption definition is not met for normalized concepts, whenever a subsumption holds in all possible models, it must be inferred via the intensional subsumption definition. This proves that the intensional subsumption definition is extensionally complete for all normalized concepts in $\mathcal{L}_1$. □

Note that we did not discuss the need for all normalization steps in the above proof, especially the completeness of constituent classification or role merging. This is because these steps were included to make structural classification of $\mathcal{L}_2$ as complete as possible although as we will see shortly, structural subsumption inference in $\mathcal{L}_2$ is still incomplete. However, these steps are *not* required for the completeness of $\mathcal{L}_1$.[16] For normalization in $\mathcal{L}_1$, the only parts that are required are expansion, DNF conversion, canonical form conversion of role restrictions, and role restriction rewriting. The necessity of each of these normalization steps for completeness were all discussed above.

Theorem 5.8 *Subsumption inference for concepts in $\mathcal{L}_1$ using the provided classification algorithm is extensionally complete.*

We only need to show that if $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ holds then the classification algorithm must infer $A \sqsubseteq B$.

Lemma 5.4 tells us that for normalized counterparts A' and B', it must hold that $A^{\mathcal{I}} \equiv A'^{\mathcal{I}}$ and $B^{\mathcal{I}} \equiv B'^{\mathcal{I}}$. From this we know that if $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ holds then $A'^{\mathcal{I}} \subseteq B'^{\mathcal{I}}$ must also hold. Then, from the correspondence between the intensional subsumption definition and the classification algorithm from theorem 5.2, we can infer that classification is extensionally complete for normalized concepts. This tells us that we must be able to infer $A' \sqsubseteq B'$ if $A'^{\mathcal{I}} \subseteq B'^{\mathcal{I}}$ holds. Furthermore, given that we maintain a syntactic mapping between concepts and their normalized counterparts, from $A' \sqsubseteq B'$, we can easily infer $A \sqsubseteq B$ using this mapping and the property given by lemma 5.4. Thus, if $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ holds, we can see that $A \sqsubseteq B$ is a valid inference that must be given by the classification algorithm. This completes the proof that classification is extensionally complete for all concepts in $\mathcal{L}_1$. □

It is interesting to note that by proving extensional soundness and completeness of the classification in theorems 5.6 and 5.8 respectively, we have effectively showed the following equivalence: *Extensional subsumption definition for $\mathcal{L}_1$ ⇔ Intensional subsumption definition for $\mathcal{L}_1$ ⇔ Subsumptions inferred via classification for $\mathcal{L}_1$*. Perhaps the most remarkable part of this equivalence is the efficiency and elegance with which the structural classification algorithm implements the other two definitions while performing classification into an entire taxonomy.

Incompleteness for classification in $\mathcal{L}_2$

Theorem 5.9 *Subsumption inference for concepts in $\mathcal{L}_2$ using the provided normalization and classification algorithm is extensionally incomplete.*

[16] In general, these extra normalization steps can be removed if subsumption is only required for $\mathcal{L}_1$ but we left them in the general algorithm for simplicity of presentation and under the assumption that most implementers will want to use $\mathcal{L}_2$ since inference is still complete for the $\mathcal{L}_1$ subset.

The language $\mathcal{L}_2$ only adds the role restrictions $\leqslant nR$, $\leqslant nR.C$, and $\forall R.C$ but unfortunately these tend to interact with the role restrictions already in $\mathcal{L}_1$ to produce subsumption cases that are not structurally identifiable. Here we give a few counterexamples to completeness of the structural algorithm and proceed to discuss why structural comparison alone cannot remedy these problems:

- $(\forall R.C_1 \sqcap \exists R.C_2) \sqsubseteq \exists R.C_1$: Obviously none of the intensional subsumption rules accounts for this subsumption case. However, this subsumption is actually valid due to the interaction of the conjoined $\forall$ and $\exists$ role restrictions. Specifically, $(forall R.C_1 \sqcap \exists R.C_2) \rightarrow \exists R.(C_1 \sqcap C_2)$. From this implied role restriction it would be easy to structurally infer the above subsumption. While it may seem that this problem could be resolved simply by elaborating the normalization algorithm with rules for inferring implied role restrictions, there are an infinite number of increasingly more subtle cases where implications like this could occur. Consequently, such role restriction inference would require full theorem-proving or equivalently powerful inference such as satisfiability testing and this would clearly obviate any claims of tractability that we would expect to achieve with a structural classification algorithm.

- $(\forall R.C \sqcap \leqslant 0R.C \sqcap \exists R.C) \equiv \bot$: This is a slightly more complex case of interaction between three role restrictions. Here $(\forall R.C \sqcap \leqslant 0R.C) \rightarrow \forall R.\bot$ which is inconsistent with $\exists R.C$ causing mutual subsumption to be inferred with $\bot$. Again, one can construct increasingly more complex cases for such role restriction interactions and there is no general method other than theorem-proving, satisfiability-testing, or some other computationally equivalent mechanism for performing complete inference in $\mathcal{L}_2$.

Consequently, we have shown two counterexamples to completeness of the structural classification algorithm thus demonstrating that the algorithm is incomplete for $\mathcal{L}_2$. □

However, we should generally point out that although such incompleteness does exist, it represents a number of fringe cases for subsumption reasoning that we expect to occur rarely on the Semantic Web. I.e., it is rather difficult to construct natural concepts according to the above definition that could not be better expressed with a more salient structure. Consequently, we expect that the majority of important subsumption relationships will stem from structural relations between concepts and thus we argue that the types of subsumptions for which structural inference in $\mathcal{L}_2$ is incomplete are unlikely to occur in practice.

This, this incompleteness does not necessarily pose a threat to the utility of structural classification algorithms for languages such as $\mathcal{L}_2$. Furthermore, given the computational efficiency gains of structural classification algorithms, it seems a small tradeoff to lose a few fringe subsumption inferences in exchange for computational tractability.

5.2 Time and space complexity

In this section, we prove some time and space complexity properties of the classification and normalization algorithms.

Time complexity of normalized concept classification

First, we will examine the time complexity of normalized concept classification with respect to the overall size of the taxonomy. Note that when we refer to the size of the taxonomy, we refer to the combined size of all normalized concepts currently classified in the taxonomy where the size of a concept is measured in the sum of all concepts, relations, and constructors used in that concept.

We assume for this analysis that the concept is already in normalized form and thus that normalization itself simply performs the task of classifying role restriction referents while taking additional time only linear in the size of the concept.[17]

Theorem 5.10 *Classification of a normalized concept in $\mathcal{L}_1$ or $\mathcal{L}_2$ runs in polynomial time in the total size of the normalized concepts in the taxonomy (including the normalized concept being classified).*

It should be apparent that once a concept is normalized, classification of a concept simply requires performing MSS, MGS, and link maintenance. For search in the MSS and MGS algorithms, the constituents of the target concept may be compared to the constituents of all concepts in the taxonomy in the worst case. We can bound the search for all concepts by a constant times the number of concepts plus the worst-case number of links between them: $c_1 \cdot (|kb| + |kb|^2)$. And we can bound the number of constituents of a concept by a constant times the size of the kb, i.e. $c_2 \cdot |kb|$. Consequently, simplifying the product of these expressions and performing asymptotic analysis, we can bound the time for MSS and MGS by $O(|kb|^3)$. Furthermore, link maintenance only requires at most $c_4 \cdot |kb|^2$ operations (i.e. no more than the total number of links in the kb). Consequently, summing this with the previous time bound for MSS and MGS and performing asymptotic analysis still yields $O(|kb|^3)$ time complexity for a single classification pass.

However, this only covers classification of the top-level concept and does not consider the fact that all of its constituents must be recursively classified. As stated above, we are assuming that since the concept has already been normalized, normalization takes only linear time aside from that required for recursive classification of role referents. Consequently, if we can show that at most a polynomial number of recursive classifications occur, than we can show that normalized concept classification is overall polynomial.

To bound the number of possible concepts that need to be recursively classified, we note that we can build a tree with each concept representing a node and children representing constituents of that node. Clearly at the leaves we have only primitive concepts which are bounded by the length of the normalized concept. And at each non-leaf node we have a composite concept or role restriction that consumes some

[17] The fact that normalization only takes linear time on an already normalized concept outside of the time required for subclassification should be obvious from inspection of Algorithm 3.

number of the children below it (note this is strictly a tree and not a DAG). Thus, at most we have to classify all of the non-leaf nodes and this is proportional to the number of leaf nodes. Since the number of leaf nodes are linear in the size of the normalized concept, we must perform at most a linear number of recursive classifications in the size of the normalized concept which is in the worst case proportional to the total size of the knowledge base.[18]

Thus, at $O(|kb|^3)$ per classification of a normalized concept and performing at most $O(|kb|)$ classifications, we can bound the total cost of classification by $O(|kb|^4)$ constant time operations. [19] Thus, classification of a normalized concept runs in time polynomial in the size of the knowledge base. □

Space complexity of concept normalization

We know that classification of a normalized concept runs in polynomial time in the total size of the normalized knowledge base, but this leaves open the question of the space complexity of normalization. That is, is a normalized concept's size bounded to be polynomial in the size of the concept?

The immediate answer is no, and this stems from the fact that the DNF expansion of a concept is known to have worst-case exponential behavior. However, we make the claim below that for the expected class of concept structures that we will likely see on the Semantic Web, such worst-case exponential blow-ups do not occur.

Hypothesis 5.11 *Normalization of a concept in $\mathcal{L}_1$ or $\mathcal{L}_2$ takes expected-case polynomial space in the size of the original concept for the expected class of concept structures that we expect to encounter on the Semantic Web.*

First, let us quickly show that other than DNF expansion, normalization takes linear space in the size of the concept: Normalization has a total of 11 steps aside from DNF expansion. An insepection of each of these 11 steps demonstrates that some of these steps shrink the size of the concept while others do not affect the size at all. Consequently, the only source of a non-polynomial expansion in size can come from the DNF expansion.

Thus, let us examine the space properties of DNF expansion: The case where DNF conversion will require exponential space to convert a formula from its given form to its normalized form is for a conjunction of several disjuncts. For example, the DNF conversion of $(A_1 \sqcup B_1) \sqcap (A_2 \sqcup B_2)$ yields $(A_1 \sqcap A_2) \sqcup (A_1 \sqcap B_2) \sqcup (B_1 \sqcap A_2) \sqcup (B_2 \sqcap B_2)$. In general for n conjuncts in the original form, the normalized form will yield 2^n disjuncts.

We claim that such a concept structure is rare because disjoined concepts usually derive their constituents from heterogeneous knowledge bases and thus are likely to use incompatible conjunctive restrictions. Thus, the conjunctive restrictions are usually associated inside the disjunction as in $((Pencil_1 \sqcap \exists.color_1.Red_1) \sqcup (Pencil_2 \sqcap \exists.color_2.Red_2))$ rather than outside the disjunction as in $(Pencil_1 \sqcup Pencil_2) \sqcap (\exists.color_2.Red_2 \sqcup \exists.color_2.Red_2)$.[20] While the latter definition could be used, in practice it seems that concepts and restrictions from similar KB's will be composed with conjunction to achieve the right level of concept specificity, and these conjoined concepts will then be disjoined at a higher level to group similar objects from heterogeneous KB's. Such organizational principles lend these concepts a natural format that is very similar to DNF therefore tending to yield polynomial conversion space in the expected case. While this is by no means a proof, we have not found sufficient evidence to the contrary from our empirical experiences working with knowledge bases on the Semantic Web. However, is such evidence existed, it would clearly require a revision of the assumptions here.

Thus, based on our assumptions that the expected type of concept structure found on the Semantic Web will likely already exhibit a DNF-like structure, it follows that the normalization of such concepts will occur in polynomial space. ◊

Time complexity of knowledge base classification

Given the previously defined theorems and hypotheses, we now state the overall time complexity result for the classification algorithm given in this paper.

Theorem 5.12 *If Hypothesis 5.11 holds for a given knowledge base in $\mathcal{L}_1$ or $\mathcal{L}_2$, then taxonomic classification of all concepts in that knowledge base runs in time polynomial to the size of the original (unnormalized) knowledge base.*

The proof of this theorem is relatively straightforward based on the implications of theorem 5.10 and hypothesis 5.11. From theorem 5.10, we know that classification of a normalized concept runs in polynomial time in the normalized size of the kb. And under the assumptions required for hypothesis 5.11, we know that the size of a normalized concept is polynomial in the size of the original concept. From this we can easily infer that the normalized kb (including the concept being classified) requires space polynomial in the size of the original kb. Thus, under the assumptions of hypothesis 5.11, classification of a *concept* (i.e. not kb) requires polynomial time in the size of the original kb.

From this previous result for the time complexity of classification of a single concept, let us now infer the time complexity for classification of an entire kb: Let us denote the polynomial time to classify a concept in a kb of size s as $p(s)$. Then it is trivial to show that classification of a number of concepts proportional to $|kb|$ is simply $p(1) + p(2) + \ldots + p(c \cdot |kb|)$. Clearly this sum is still polynomial in the size of the original kb and thus under the assumptions of hypothesis 5.11, taxonomic classification of all concepts in that knowledge base runs in time polynomial to the size of the original kb. □

Finally we note that this result along with the expressivity of languages $\mathcal{L}_1$ and $\mathcal{L}_2$ is precisely what was required in

[18] This would only happen though if the concept being classified happened to dwarf the size of all other concepts in the knowledge base.

[19] Note that the classification algorithm is typically orders of magnitude much more efficient than this but our only goal here is to show that classification is polynomial. A more detailed algorithmic analysis would involve a substantially greater amount of derivation. However, empirically, algorithms like the one presented here tend to run in quadratic time in the size of the knowledge base.

[20] The subscripts 1 and 2 indicate the KB from which the concepts and roles were drawn.

Section 1.3.

6 Conclusion

Before we proceed with our conclusions, we should reiterate that we are not arguing for a general purpose subsumption algorithm to tackle all description logic problems. Rather, we are identifying an important use of description logics in the context of reasoning over distributed knowledge bases on the Semantic Web, identifying the major sources of intractability for such reasoning, and fine tuning an algorithm based on this analysis to achieve expected case polynomial-time performance in the size of the kb. Since a sound and complete algorithm for a language as expressive as $\mathcal{L}_1$ is likely NP-Complete (although no formal complexity result has been given for $\mathcal{L}_1$), in some sense, adapting and optimizing an algorithm to its expected use cases is perhaps the best that can be hoped for. Furthermore, as we did for $\mathcal{L}_2$, we believe that it is desirable to allow some incompleteness to avoid exponential complexity cases if one can show that this is relatively benign. We belive this is far superior to working with an algorithm for a less expressive language or using an algorithm that is likely to exceed a reasonable running time on a large knowledge base. Thus, it is our opinion that such tradeoffs as described in this paper must be taken in order to achieve scalable and practical description logic classification algorithms in practice.

6.1 Summary

In summary, let us review the basic motivations behind this work and the results presented here.

In reasoning on the Semantic Web, it is likely that as knowledge bases grow and make use of terms defined in other knowledge bases, a large source of logically composed description logic concepts will begin to emerge. Consequently, since these composed concepts make use of common primitives, it will likely be useful to have some means of reasoning about subsumption relationships between such concepts.

Autonomous taxonomic classification as pioneered in the field of description logics seems to pose an elegant solution to this problem with the one caveat of computational intractability. Specifically, for the language expressiveness needed for representation on the Semantic Web (namely conjunctive and disjunctive composition as well as many commonly used role restrictions), it is likely impossible to achieve a sound and extensionally complete, polynomial-time inference algorithm. Yet, given the size of the Semantic Web, a polynomial-time inference algorithm is almost a necessity, soundness is clearly a necessity, and completeness is generally desired unless we can show that the sources of incompleteness in an algorithm are benign.

Additionally, there is one other concern with reasoning on the Semantic Web that underscores the limitations of current approaches and is perhaps one of the most important points in this paper. As previously argued by Woods, *the primary tractability concern in description logic reasoning over large databases is not the cost of subsumption, but rather the cost of classifying into a large taxonomy*. That is, efficient subsumption techniques alone do not guarantee efficient taxonomic classification and yet the latter should be our primary goal for most real-world problems. Thus, in order to satisfy the above requirements of the Semantic Web while providing an algorithm that focuses on efficient taxonomic classification, we propose the following solution:

Intensional definitions of subsumption have traditionally lent themselves to efficient taxonomic classification via structural comparison. However, the drawback to these approaches has traditionally been that they could not handle expressive languages. Consequently, we have relied on three arguments to make the case that intensional subsumption approaches can indeed provide the reasoning capabilities that we require for the Semantic Web:

- There are certain language constructs that are rarely used and which can be omitted from consideration. For example, axioms between two structured concepts (e.g. $A \sqcup B \sqsubseteq C \sqcap D$) are rare in practice and can be omitted without a practical loss in expressiveness.
- While inference in more expressive description languages is incomplete, this incompleteness stems from subsumptions that are often infrequent in practice and therefore overall relatively benign.
- While in the general case, we cannot omit the possibility of any valid concept structure, in practice, we expect to encounter concepts that are fairly similar to disjunctive normal form in compositional structure. This means that normalization of these concepts to DNF is likely to be possible in polynomial space. Since we can show that DNF conversion is the only source of exponential algorithm behavior for structural subsumption in $\mathcal{L}_1$, we can show that under the assumption of polynomial-space DNF conversion, classification will run in polynomial-time in the size of the knowledge base for $\mathcal{L}_1$.

With the above assumptions, we then proceeded to extend traditional intensional subsumption definitions to support both conjunction and disjunction. We provided a novel structural subsumption algorithm to handle two languages ($\mathcal{L}_1$ and $\mathcal{L}_2$) of differing expressivity, the latter providing a major subset of the DAML+OIL language used on the Semantic Web. We then proved soundness of the classification algorithm for $\mathcal{L}_1$ and $\mathcal{L}_2$ and completeness for inference in $\mathcal{L}_1$. We showed that the application of the inference algorithm applied to $\mathcal{L}_2$ yielded incompleteness, but was relatively benign with respect to practical reasoning as argued above. Finally, we showed that if DNF normalization of a concept could be performed in polynomial space then the overall algorithm required time polynomial in the size of the taxonomy.

6.2 Future work

The previous decade of description logic research has seen little focus on structural subsumption techniques likely due to the traditional perception of its inability to handle expressive languages in an extensionally sound and complete manner. However, this work has intended to challenge some of these perceptions by showing that structural subsumption could be extended to handle sound and complete subsumption for conjunctive and disjunctive definitions and a reasonable set of

role restrictions. Additionally this paper has intended to support the notion that some expressiveness (i.e. complement) is not practically useful for some applications and some incompleteness can be considered relatively benign (i.e. as for $\mathcal{L}_2$). Such ideas challenge previous beliefs about structural classification approaches thus paving the way for future research in taxonomic classification as opposed to the recent focus on subsumption testing between individual concepts.

One important question for future research is what additional expressiveness can be handled via an intensional subsumption definition and an associated structural classification algorithm? For example, could an augmented version of the above algorithm allow for extensionally sound and complete subsumption in $\mathcal{L}_1$ augmented with full complement? Or is better to only allow complement of primitive concepts? Also, what additional restriction constructors or role constructors can be added to the algorithm? For example, how easy would it be to augment $\mathcal{L}_1$ or $\mathcal{L}_2$ with instance specific role restrictions such as $fills$? Such role restrictions prove quite useful in practice but would require an extension to the previously given intensional subsumption definition as well as an analysis of the completeness of such a definition with respect to the other constructors in $\mathcal{L}_1$ or $\mathcal{L}_2$.

Another important question for future research is that of improving the efficiency of classification algorithms. For example, for the language $\mathcal{L}_1$ minus disjunction, research in the Conceptual Indexing group at Sun Microsystems Research Labs has yielded structural classification algorithms that are orders of magnitude more efficient than the basic algorithms presented here (see [Woods, 1997] for an introduction to this research domain). The optimizations in these algorithms allow much of the search to be pruned based on provable techniques for constraining search. Furthermore, even more aggressive partitioning techniques may allow us to prune many more subsumption tests. Consequently, while the algorithm presented here was chosen for simplicity, there is clearly much potential for further optimization of this algorithm that can yield speedups of a few orders of magnitude.

At the very least, future work along both of these lines will be a necessity to ensure that description logic classification algorithms can practically apply and tractably scale to the many applications of description logic reasoning that will likely emerge as the Semantic Web matures.

7 Software

A limited implementation of the algorithms discussed here has been implemented and integrated with JTP [Frank, 1999], a Java-based theorem prover, developed at the Stanford Universiry Knowledge Systems Lab. The following sections indicate how to obtain the software, its limitations compared to the full algorithm given in this paper, directions for using the subsumption reasoner from the JTP command line or integrating it with other software, and finally a sample application demonstrating its usage.

7.1 Obtaining the software

The latest release of the JTP software which includes a special purpose DAML+OIL taxonomic reasoner based on the ideas presented here can be downloaded from `http://www.ksl.stanford.edu/software/JTP/`. Instructions are available from this site for how to install JTP and run it. To determine that the software and taxonomic reasoner are running correctly, one can compare the sample application output at the end of this section with the same commands executed on the reasoner under test.

7.2 Implementation discussion

The implementation provided in JTP differs slightly from that presented here. It is implemented for $\mathcal{L}_2$ which means that it does have some incompleteness as discussed previously. However this implementation additionally lacks two features which lead to additional incompleteness:

1. This software does not implement definition expansion or DNF conversion although it does implement conversion of role restrictions to canonical form. This means that structural subsumption is applied directly to the logical concept structure as it is was defined when it was created. Consequently two concepts that would normalize to the same structure are not guaranteed to be found as mutual subsumers.
2. This software does not implement all of the intensional subsumption rules as defined in Table 2. Specifically, it does not implement all subsumption tests between primitive and conjunctive or disjunctive concepts and it does not implement the subsumption test between conjunctive and disjunctive concepts (this latter subsumption occurs so rarely in practice that it can be safely ignored).

In the future, we hope to add full normalization and all intensional subsumption rules to make the reasoner complete for $\mathcal{L}_1$. Then it will only be incomplete for $\mathcal{L}_2$ w.r.t. the previous section's discussion of incompleteness for the intensional subsumption rules defined in Table 2.

7.3 Code overview

The Java code for the JTP classifier can be broken down into two distinct portions: a JTP specific interface located in the JTP package `jtp.frame.classifier` and a general description logic classifier located in `dl.classifier` which is imported from `jtp\lib\dl.jar`. We will discuss each of these code portions in more detail:

- `jtp.frame.classifier`: This package contains two classes which implement the *telling* and *asking* interfaces for the description logic special purpose reasoner. *ClassifierTellingReasoner* simply caches all DAML+OIL RDF statements as JTP broadcasts the statements to registered telling listeners. It also implements an undo/redo mechanism for restoring the cache to a given checkpoint state when requested by JTP. *ClassifierAskingReasoner* is where the bulk of the interface occurs and performs three main tasks: First, it answers JTP queries concerning *daml:subClassOf* and *daml:disjointWith* statements. Second, whenever a query is made, the reasoner attempts to recursively construct and classify all classes and restrictions whose defi-

nition is complete.[21] This involves retrieving all relevant RDF statements for a given class, restriction, or relation, and notifying the classifier in the `dl` package of the structure and names for these new objects. Once these objects are added to the classifier, the classifier automatically classifies them. Third, this class also implements an undo/redo mechanism that maintains and restores the classifier state when requested by JTP.

- `dl.classifier`: This package implements a classifier for language $\mathcal{L}_2$ that is independent of the DAML+OIL syntax. This is why the `jtp.frame.classifier` must take care of all translation between JTP and the classifier. The classifier itself is quite complex and will not be discussed in its entirety here. However there are two main classes that one should be familiar with. *KbInterface* implements a Java *String*-based interface to the classifier. This class is used exclusively by *ClassifierAskingReasoner* for both adding concepts and restrictions to the classifier and for the querying the classifier's taxonomy structure. *Classifier* implements all of the classification algorithms and is basically equivalent to a direct implementation of Algorithms 1-5 with the exception of the afore-mentioned missing normalization steps. It also implements all required helper functions outlined in section 4.4. All other classes in this package implement basic functionality for the classifier and should not require modification.

7.4 Command line interface

Since the taxonomic reasoner interacts with JTP as a backchaining reasoner, it will be dispatched any time a subsumption or disjointness query is made. [22] Consequently, one can invoke the taxonomic reasoner simply by asking a *daml:subClassOf* or *daml:disjointWith* question from the JTP command line interface.

For a more versatile interface, one can invoke a text-based taxonomy browser from the JTP command line interface by using the `tax` command. While there is an extensive set of commands for viewing the taxonomy structure, the following commands are likely to prove most useful:

- `hg <conc-name> <depth>`: This is perhaps the most useful command which allows one to browse the taxonomy in an hourglass format (i.e. centered on a provided concept and showing the parents above and children below to the given depth). In the output, `...` indicate that more concepts exist but are not being displayed since the depth limit has been exceeded. And `[*]` indicates that a concept has already been seen and its parent or child structure will be truncated since it has been printed elsewhere. Note that in text form, TOP stands for $\top$, BOTTOM for $\bot$, and MOD for $\top_{rel}$. See the applications section for sample commands and output from applying this to a web-accessible DAML+OIL kb.
- `showc c <none|direct|all> <conc-name>` : This command allows one to view the detailed structure of a concept.
- `showc m <none|direct|all> <conc-name>` : This command allows one to view the detailed structure of a modifier including it's type, relation, and concept referent.
- `listc <none|direct|all>`: This command allows one to browse all concept names that have been loaded into the taxonomy. This is useful for finding a concept name once a kb has been loaded.
- `help`: This command gives an extensive help printout on all commands available from the taxonomy broswer.
- `exit`: This command exits the taxonomy broswer and returns control to JTP's command line interface.

7.5 Integration with other software

Figure 5 provides a brief introduction to the Taxonomy API in JTP. And figure 6 provides sample code for using the Taxonomy API.

7.6 Sample applications

Figure 7 provides an excerpt of the *Dogs kb* used in the following examples. Figures 8 and 9 demonstrate the application of JTP's taxonomic reasoning facility to a sample web accessible DAML+OIL knowledge base. This application demonstrates use of the taxonomy browser on a kb with a wide variety of subsuming concept structures. Such a structure could also be retrieved directly through the Taxonomy API discussed in figures 5 and 6.

Acknowledgments

The author would like to thank Bill Woods for providing the foundational motivations for subsumption, taxonomy, and the design of classification algorithms; *none* of this work could have been accomplished without the knowledge acquired from many summer internships working for Bill at Sun Microsystems Research Labs. The author would also like to thank Richard Fikes for his guidance, support, and suggestions throughout the completion of this work. This work was primarily completed at and funded by the Knowledge Systems Lab (KSL) at Stanford University.

References

[Bemers-Lee *et al.*, 2001] Tim Bemers-Lee, Jim Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[21] Since a DAML+OIL knowledge base is read one RDF statement at a time rather than as a whole, it is possible that a definition may not be complete when a query is made. Consequently the reasoner takes a lazy approach to class and restriction building by adding all complete definitions to the classifier at query-time. Incomplete definitions are not compiled into their respective classes or restrictions and are checked on each subsequent query until their definition is complete.

[22] Subsumption is invoked for disjointness reasoning since taxonomic structure can be used to determine whether the superclasses of any two concepts have a disjointness relationship asserted. Note however that this type of disjointness reasoning is very simple and is extensionally incomplete.

[Borgida *et al.*, 1989] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. CLASSIC: a structural data model for objects. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 58–67, Portland, Oregon, 1989.

[Brachman *et al.*, 1991] Ronald J. Brachman, Deborah L. McGuiness, Peter F. Patel-Schneider, and Lori A. Resnick. Living with CLASSIC: when and how to use a KL-ONE-like language. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, San Mateo, US, 1991.

[Donini, 2002] Francesco M. Donini. Complexity of reasoning. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, chapter 3. Cambridge University Press, 2002.

[Frank, 1999] Gleb Frank. A general interface for interaction of special-purpose reasoners within a modular reasoning system. In *Papers from the AAAI Fall Symposium*, pages 57–62, 1999.

[Horrocks *et al.*, 2000] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–264, 2000.

[Horrocks *et al.*, 2001] Ian Horrocks, Frank van Harmelen, and Peter Patel-Schneider. DAML+OIL language specification, March 2001. Document located on-line at http://www.daml.org/2001/03/daml+oil-index.html.

[Levesque and Brachman, 1985] H. J. Levesque and R. J. Brachman. A fundamental tradeoff in knowledge representation and reasoning (revised version). In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, pages 41–70. Morgan-Kaufmann, Inc., 1985.

[Woods and Schmolze, 1992] William A. Woods and James G. Schmolze. The KL-ONE family. *Semantic Networks in Artificial Intelligence*, 23(2-5):133–178, 1992. Published as a special issue of Computers & Mathematics with Applications.

[Woods, 1991] William A. Woods. Understanding subsumption and taxonomy: A framework for progress. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, San Mateo, US, 1991.

[Woods, 1997] William A. Woods. Conceptual indexing: A better way to organize knowledge. Technical report, Sun Microsystems Laboratories, 1997.

The core of the description logic utilities are located in the package 'dl.jar'. To interface with the taxonomy API in Java, you'll first need to get the classification kb itself which is an instance of dl.classifier.Kb. This is 'public static' in the JTP interface so you can access it via the following code:

```
import dl.classifier.*; // The source of class 'Kb'
import jtp.frame.classifier.*; // The source of class
                               // 'ClassifierAskingReasoner'

...

Kb _myKb = ClassifierAskingReasoner._ClassifierKb;
```

Once you have an instance of Kb, it is important to realize that the dl utilities use the class dl.classifier.Concept as the primary object for talking about DAML classes. So, everything will need to be translated to and from dl.classifier.Concept in order to interact directly with the taxonomy.

To get a Concept from a String 'str':

```
// Note that str has be in the same format used in the taxonomy
// browser ("kb#::classname").  So for example,
// "http://dogs.com/dogs.daml#::Dog"
Concept c = Concept.CastToConcept(Resource.GetResource(str));
```

To get a Concept from a JTP symbol 'sym':

```
Concept c = Concept.CastToConcept(Resource.GetResource(
   ClassifierAskingReasoner.ConvertSymObjectToString(sym)));
```

To get the string from a Concept 'conc':

```
String s = conc.toString();
```

Now that we have the classification kb as well as the methods for Concept translation, one can use the following interface method to access the taxonomy links:

```
java.util.Set s = _myKb.getTransitiveLinkClosure(conc, rel,
                                 link_type, direct_only);
```

where the parameters are the following:

```
conc: (Concept) The root concept whose links we are looking at

rel:  (Relation) Either Relation._KIND_OF (subsumers) or
              Relation._INV_KIND_OF (subsumees)

link_type: (int) A bitwise OR of one or more of the following:
              Kb._A_LINK - links to axiomatic subsumers/subsumees
              Kb._I_LINK - links to or from conjunctive constituents
              Kb._D_LINK - links to or from disjunctive constituents
              Kb._S_LINK - links to structural subsumers/subsumees
              Kb._R_LINK - redundant links to or from conjunctive
                           constituents

direct_only: (boolean) Do we want just direct links (true), or the
              full transitive closure (false)
```

Note that the above link names are slightly different than those referenced in the paper. For all intents and purposes, *I* and *R* links should be considered equivalent to the *C* links discussed in the paper.

Figure 5: Introduction to the Taxonomy API

Following is some sample code to print out the direct subsumers and all subsumees of a 'Dog' concept that has been loaded into JTP:

```
// Assuming dl.classifier.*, jtp.frame.classifier.*, and java.util.*
// have been imported.

// Get the taxonomy
Kb _myKb = ClassifierAskingReasoner._ClassifierKb;

// Get a concept object for the following DAML class
Concept myConc = Concept.CastToConcept(
   Resource.GetResource("http://dogs.com/dogs.daml#::Dog"));


// Get a set of the direct subsumers of myConc
Set dir = _myKb.getTransitiveLinkClosure(myConc, Relation._KIND_OF,
                        Kb._A_LINK | Kb._I_LINK | Kb._D_LINK |
                        Kb._S_LINK | Kb._R_LINK, true);

// Get a set of *all* subsumees of myConc
Set all = _myKb.getTransitiveLinkClosure(myConc, Relation._INV_KIND_OF,
                        Kb._A_LINK | Kb._I_LINK | Kb._D_LINK |
                        Kb._S_LINK | Kb._R_LINK, false);

// Print out the answers
Iterator i = dir.iterator();
System.out.println("Displaying direct subsumers of Dog:");
while (i.hasNext()) {
   System.out.println(((Concept)i.next()).toString() + " ");
}

i = all.iterator();
System.out.println("Displaying all subsumees of Dog:");
while (i.hasNext()) {
   System.out.println(((Concept)i.next()).toString() + " ");
}
```

Figure 6: Sample usage of Taxonomy API

Source kb: http://www.ksl.stanford.edu/people/sscott/dogs.daml

```
<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  ...
  xmlns     ="http://www.ksl.stanford.edu/people/sscott/dogs.daml#"
>

<!-- =========== Primitive class/relation definitions ========== -->

<daml:Class rdf:ID="Animal"> </daml:Class>

<daml:Class rdf:ID="Dog">
  <rdfs:subClassOf   rdf:resource="#Animal"/>
  <daml:disjointWith rdf:resource="#Cat"/>
</daml:Class>

<daml:ObjectProperty rdf:ID="with">
  <rdfs:subPropertyOf rdf:resource="#mod"/>
</daml:ObjectProperty>

...

<!-- =========== Composite class definitions ========== -->


<daml:Class rdf:ID="BlackSpots">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Spots"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#mod"/>
      <daml:hasClass rdf:resource="#Black"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="DogWithMinFiveBlackSpots">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Dog"/>
    <daml:Restriction daml:minCardinalityQ="5">
      <daml:onProperty rdf:resource="#with"/>
      <daml:hasClassQ rdf:resource="#BlackSpots"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="DogOrBrownFurOrBlackFur">
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Dog"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#with"/>
      <daml:hasClass rdf:resource="#BrownFur"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#with"/>
      <daml:hasClass rdf:resource="#BlackFur"/>
    </daml:Restriction>
  </daml:unionOf>
</daml:Class>

...

</rdf:RDF>
```

Figure 7: Excerpt from the DAML+OIL *Dogs kb* used in the following examples.

Taxonomy from *Dogs Kb* centered at 'Dog' (note use of different restrictions - existential, qualified cardinality, etc...).

Source kb: http://www.ksl.stanford.edu/people/sscott/dogs.daml

Command: hg http://www.ksl.stanford.edu/people/sscott/dogs.daml#::Dog

```
          |-  TOP
      |- http://.../dogs.daml#::Thing
      |   |-  TOP  [*]
      |- http://.../dogs.daml#::AnimalOrDarkFur
  |- http://.../dogs.daml#::Animal
  |           |- ...
  |       |- http://.../dogs.daml#::AnimalOrDarkFur [*]
  |   |- http://.../dogs.daml#::DogOrBrownFurOrBlackFur
  |- http://.../dogs.daml#::DogOrBrownFur
  |   |- ...
  |- http://.../dogs.daml#::DogOrBrownFurOrBlackFur [*]
http://.../dogs.daml#::Dog
  |- http://.../dogs.daml#::LittleDog
  |   |- http://.../dogs.daml#::LittleDogModOld
  |       |- http://.../dogs.daml#::DogModOldModLittle
  |           |- http://.../dogs.daml#::LittleDogModOld [*]
  |               |- ...
  |- http://.../dogs.daml#::BigDog
  |   |- http://.../dogs.daml#::BigDogWithDarkFur
  |       |- http://.../dogs.daml#::BigDogWithBrownFur
  |           |-  BOTTOM
  |- http://.../dogs.daml#::DogWithSpots
  |   |- http://.../dogs.daml#::DogWithMinFiveSpots
  |   |   |- http://.../dogs.daml#::DogWithMinFiveBlackSpots
  |   |       |-  BOTTOM  [*]
  |   |- http://.../dogs.daml#::DogWithDarkSpots
  |       |- http://.../dogs.daml#::DogWithBrownSpots
  |       |   |-  BOTTOM  [*]
  |       |- http://.../dogs.daml#::DogWithBlackSpots
  |           |- http://.../dogs.daml#::DogWithMinFiveBlackSpots [*]
  |               |- ...
  |- http://.../dogs.daml#::DogWithFur
      |- http://.../dogs.daml#::DogWithDarkFur
          |- http://.../dogs.daml#::BigDogWithDarkFur [*]
              |- ...
```

Figure 8: DAML+OIL taxonomy generated by the JTP taxonomy browser.

Taxonomy from *Dogs Kb* centered at 'BigDog' (note the mix of disjunctive and conjunctive concept structure).

Source kb: http://www.ksl.stanford.edu/people/sscott/dogs.daml

Command: hg http://www.ksl.stanford.edu/people/sscott/dogs.daml#::BigDog

```
             |-  TOP
         |- http://.../dogs.daml#::Thing
         |   |-  TOP  [*]
         |- http://.../dogs.daml#::AnimalOrDarkFur
     |- http://.../dogs.daml#::Animal
     |       |   |- ...
     |       |- http://.../dogs.daml#::AnimalOrDarkFur [*]
     |   |- http://.../dogs.daml#::DogOrBrownFurOrBlackFur
     |- http://.../dogs.daml#::DogOrBrownFur
     |   |- ...
     |- http://.../dogs.daml#::DogOrBrownFurOrBlackFur [*]
 |- http://.../dogs.daml#::Dog
 |   |- [_GEN_ <_MOD_>]
 |- [_EXISTS_ <http://.../dogs.daml#::mod,
 |             http://.../dogs.daml#::Big,1>]
 |       |- ...
 |   |- http://.../dogs.daml#::DogOrBrownFur [*]
 |- http://.../dogs.daml#::BigDogOrBrownFur
http://.../dogs.daml#::BigDog
 |- http://.../dogs.daml#::BigDogWithDarkFur
     |- http://.../dogs.daml#::BigDogWithBrownFur
         |-  BOTTOM
```

Figure 9: Another DAML+OIL taxonomy generated by the JTP taxonomy browser.