Computing Least Common Subsumers in Description Logics

William W. Cohen
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
wcohen@research.att.com

Alex Borgida*
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08903
borgida@cs.rutgers.edu

Haym Hirsh
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08903
hirsh@cs.rutgers.edu

Abstract

Description logics are a popular formalism for knowledge representation and reasoning. This paper introduces a new operation for description logics: computing the "least common subsumer" of a pair of descriptions. This operation computes the largest set of commonalities between two descriptions. After arguing for the usefulness of this operation, we analyze it by relating computation of the least common subsumer to the well-understood problem of testing subsumption; a close connection is shown in the restricted case of "structural subsumption". We also present a method for computing the least common subsumer of "attribute chain equalities", and analyze the tractability of computing the least common subsumer of a set of descriptions—an important operation in inductive learning.

Introduction and Motivation

Description logics (DLs) or *terminological logics* are a family of knowledge representation and reasoning systems that have found applications in several diverse areas, ranging from database interfaces [Beck *et al.*, 1989], to software information bases [Devanbu *et al.*, 1991] to financial management [Mays *et al.*, 1987]. They have also received considerable attention from the research community (e.g., [Woods and Schmolze, 1992].)

DLs are to used to reason about *descriptions*, which describe sets of atomic elements called *individuals*. Individuals can be organized into *primitive classes*, which denote sets of individuals, and are related through binary relations called *roles* (or *attributes* when the relation is functional). For example, the individuals `Springsteen` and `BorntoRun` might be related by the `sings` role, and `Springsteen` might be an instance of the primitive class `PERSON`. *Descriptions* are composite terms that denote sets of individuals, and are built from primitive classes (such as `PERSON`), and restrictions on the properties an individual may have, such as the kinds or number of role fillers (e.g., "`persons that sing at least 5 things`"). For example, the statement "`all songs sung by Springsteen (and there are at least 5) are set in New Jersey`" could be expressed by attaching to the individual `Springsteen` the description `(AND (AT-LEAST 5 sings) (ALL sings (FILLS setting NJ)))`.

Knowledge base management systems (KBMS) based on DLs perform a number of basic operations on descriptions: for example, checking if a description is incoherent, or determining if two descriptions are disjoint. An especially important operation on descriptions is testing *subsumption*: D1 *subsumes* D2 iff it is more general than D2. Efficient implementation of such operations allows a KBMS to organize knowledge, maintain its consistency, answer queries, and recognize conditions that trigger rule firings.

This paper introduces a new operation for description logics: computing the *least common subsumer (LCS)* of a pair of concept (i.e., finding the most specific description in the infinite space of possible descriptions that subsumes a pair of concepts.[1]) This operation can also be thought of as constructing a concept that describes the largest set of commonalities between two other concepts. In logic programming, similar operations called "least general generalization" and "relative least general generalization" have been extensively studied [Frisch and Page, 1990; Plotkin, 1969; Buntine, 1988]; in the context of DLs, there are a number of circumstances in which the LCS operation is useful.

Learning from examples. Finding the least general concept that generalizes a set of examples is a common operation in inductive learning. For example, [Valiant, 1984] proved that k-CNF (a class of Boolean functions) can be probabilistically learned by computing the LCS of a set of positive examples; also, many experimental learning systems make use

*On sabbatical leave.

[1] This should not be confused with the "most specific subsumer" operation, which searches the (finite) space of *named concepts* to find the most specific named concept(s) that subsumes a single concept [Woods and Schmolze, 1992].

of LCS-like operations [Flann and Dietterich, 1989; Hirsh, 1992]. A companion paper explores the learnability of DLs via LCS operations [Cohen and Hirsh, 1992]; DLs are useful for inductive learning because they are more expressive than propositional logic, but still (in some cases) tractably learnable.

Knowledge-base "vivification" (e.g. [Borgida and Etherington, 1989; Levesque, 1986]). Reasoning with a knowledge base that contains disjunctive facts is often intractable. *Vivification* is a way of logically weakening a knowledge base to avoid disjunction: for example, rather than encoding in the knowledge base the fact that `PIANIST(Jill)` $\vee$ `ORGANIST(Jill)`, one might elect to encode some conjunctive approximation to this disjunction, such as `KEYBOARD-PLAYER(Jill)`. In DLs that reason tractably with non-disjunctive descriptions, replacing a disjunction $D_1 \vee \ldots \vee D_n$ with the LCS of $D_1, \ldots, D_n$ is a vivification operation, as it leads to an approximation that can be reasoned with effectively.

Other reasoning tasks. Frish and Page [1990] argue that certain types of abduction and analogy can also be performed using LCS operations. Also, many specific applications of LCS operations have been described. For example, Schewe [1989] proposes an approach to the important problem of developing of variants of existing software using the Meson DL; a key notion in his approach is the use of the LCS of pairs of descriptions, denoting the desired and existing portions of code respectively.

In the remainder of this paper, we first precisely define the LCS of two descriptions. We then develop an understanding of the LCS operation by analyzing the relation of LCS to the well-studied and well-understood problem of subsumption. A close connection is shown for the special case of "structural subsumption": for example, given a recursive structural subsumption algorithm and definitions of LCS for the base cases one can mechanically construct a recursive LCS algorithm. Finally, we analyze the complexity of computing the LCS of attribute chain equalities, an important construct for DLs that normally requires a non-structural subsumption algorithm.

Our results are not specific to any particular DL; however, we will present our examples in a specific DL, namely CLASSIC [Borgida *et al.*, 1989], the syntax of which is summarized in an appendix.

The `lcs` Operator

Let $\mathcal{L}$ be a description logic and let $\Longrightarrow$ be its subsumption relationship—i.e., if description D1 subsumes D2 then we will write D2$\Longrightarrow$D1. By definition, $\Longrightarrow$ must be a partial order on (denotations of) descriptions. It is possible that D1$\Longrightarrow$D2 and D2$\Longrightarrow$D1 without D1 and D2 being syntactically identical; in this case, we will say that D1 and D2 are *semantically equivalent*, and write D1$\equiv$D2. C $\in \mathcal{L}$ is a *least common subsumer (LCS)* of D1 and D2 iff a) C subsumes both D1 and D2, and b) no other common subsumer of D1 and D2 is strictly subsumed by C.

Notice that in general, a least common subsumer may not exist, or there may be many semantically different least common subsumers. We define the *lcs operator* to return a set containing exactly one representative of the equivalence class (under $\equiv$) of every least common subsumer of D1 and D2. More precisely:

Definition 1 *If D1 and D2 are concepts in a description language $\mathcal{L}$ then* `lcs`*(D1,D2) is a set of concepts* $\{\mathrm{C}_1, \ldots, \mathrm{C}_i, \ldots\}$ *such that a) each C_i is a least common subsumer of D1 and D2, b) for every C that is a least common subsumer of D1 and D2, there is some C_i in* `lcs`*(D1,D2) such that* $\mathrm{C}_i \equiv \mathrm{C}$*, and c) for* $i \neq j$, $\mathrm{C}_i \not\equiv \mathrm{C}_j$.

However, all extant DLs provide some description constructor (usually called AND) which builds a description whose denotation is the intersection of the denotation of its arguments. If such a constructor exists, then it can be shown that if any least common subsumer exists, it is unique up to equivalence under $\equiv$. We have the following proposition.

Proposition 2 *For any DL with the description intersection constructor* (AND)*, if* `lcs`*(D1,D2) is not empty, then it is a singleton set.*

(Proof by contradiction: if two incomparable LCS exist, their intersection is a more specific common subsumer, which could not be equivalent to them.) In the remainder of the paper, we will consider only DLs equipped with a constructor AND.

Relating `lcs` and Subsumption

The choice of constructors available for building descriptions clearly affects the expressive power of the DL and the complexity of reasoning with it, as well as affecting the computation of the `lcs` operator. Since much research has been devoted to determining subsumption in various DLs (especially its tractablility), it would useful to find a correlation between computing subsumption and computing `lcs`. The following two theorems show that this is not likely to be straightforward in the general case:

Theorem 3 *There exists a DL such that the subsumption problem is polynomial-time decidable but computing the* `lcs` *operator cannot be done in polynomial time.*

Proof: Consider the DL Bit_n where descriptions have the form (AND (PRIM $\langle vec \rangle$)*), where $\langle vec \rangle$ is a bit vector γ of length n, where $\gamma = 10\cdots0, \gamma = 0\cdots01$ or γ has $\frac{n}{2}$ 1's. (The following would then be a description in Bit_4: (AND (PRIM 1100) (PRIM 0101) (PRIM 1000)).) The semantics of the PRIM constructor are that it represents a primitive concept (i.e., membership in its extension is explicitly asserted) but with the constraint that (PRIM v1) must be a superset of (PRIM v2) if $v1 \wedge v2 = v1$. A simple polynomial-time algorithm for determining whether some C1 subsumes C2 is to check, for every primitive (PRIM v) appearing in C1, whether C2 has a primitive (PRIM w) such that $v \wedge w = v$. On the other hand, the computation of `lcs((AND (PRIM 10...0)), (AND (PRIM 0...01))` requires exponential time since the answer (the conjunction of all the other bit-vector primitives in the language) is of exponential size.[2] ■

Theorem 4 *There exists a DL for which the* `lcs` *operator can be computed in linear time, while subsumption is co-NP-hard.*

Proof: Consider the DL containing the constructors PRIM, AND, and OR (where OR denotes disjunction). For this language $\texttt{lcs}(C, D) = \{(\text{OR } CD)\}$ is a correct implementation. However subsumption is co-NP-hard; see, for example, [Borgida and Etherington, 1989] for a proof. (We note, however, that for this DL testing semantic equivalence is also intractable, and thus the `lcs` as we compute it is perhaps less useful for reasoning.) ■

However, although `lcs` and subsumption appear to be unrelated in the general case, they are closely related in the restricted setting described below.

Relating lcs and Structural Subsumption

In a typical DL, description constructors interact in various ways; for example, `(AND (AT-LEAST 2 sings) (AT-MOST 1 sings))` is equivalent to the inconsistent concept and is hence subsumed by any other description. Many, though not all, DLs reason by first reducing descriptions to some *normal form*, where implicit facts are explicated, inconsistencies (such as the one above) are detected, and so on. Subsumption tests and other operations are then performed on normal-form representations of the description via relatively simple algorithms—algorithms which need *not* consider the possible interactions among various constructors. In particular, subsumption testing can be done on normal-form descriptions by independently considering the parts of descriptions built with the same constructor.

We will write normalized descriptions (as opposed to CLASSIC descriptions) as first-order terms without variables, where the various concept constructors appear as functors, with integers, tuples, lists, other terms, etc, as arguments (e.g., `and([atleast(5,sings), all(sings,fills(setting,NJ))])`. In such a normal form, the subsumption algorithm[3] `Subsumes?` would return `false` on the invocation `Subsumes?(atmost(foo), atleast(bar))` because the constructors involved are different. We will call such a subsumption relationship *structural subsumption*[4]:

Definition 5 `Subsumes?` *is a* structural subsumption relationship *iff* $\texttt{Subsumes?}(k_1(\alpha), k_2(\beta))$ *is false whenever* $k_1 \neq k_2$.

The list of DLs for which structural subsumption is used include Kandor [Patel-Schneider, 1984], Krypton [Brachman *et al.*, 1983], Meson [Edelman and Owsnicki, 1986], ENTITY-SITUATION [Bergamaschi *et al.*, 1988], CLASSIC (without the SAME-AS constructor) and the logic of [Patel-Schneider, 1989]. Observe however that a structural subsumption relationship need not be tractable: it may be that putting a description into normal form is difficult, or that for some specific constructor subsumption is difficult to compute.

Given that `Subsumes?` is a structural subsumption relationship, $\texttt{Subsumes?}(k(\alpha), k(\beta))$ can depend only on the specifics of α and β. Thus a structural subsumption relationship can be fully defined by a table specifying $\texttt{Subsumes?}(k(\alpha), k(\beta))$ for the various term constructors k; this means that subsumption imposes a partial order (which we denote $\leq_k$) on the space V_k of possible arguments to constructor k. More precisely we define $\alpha \leq_k \beta$ iff $\texttt{Subsumes?}(k(\alpha), k(\beta))$.

As an example, let $\mathcal{N}$ denote the natural numbers and $\mathcal{R}ole$ the set of possible roles. For the constructor AT-MOST in CLASSIC, $V_{\text{AT-MOST}}$ is $\mathcal{N} \times \mathcal{R}ole$, and $((n_1, r_1) \leq_{\text{AT-MOST}} (n_2, r_2))$ is true iff $n_1 \leq n_2$ and $r_1 = r_2$. The `lcs` function for this constructor can be defined as follows:

$$\texttt{lcs}(\texttt{atmost}(n_1, r_1), \texttt{atmost}(n_2, r_2)) =$$
$$\text{if } r_1 = r_2 \text{ then } \texttt{atmost}(\max(n_1, n_2), r_1) \text{ else } \texttt{THING}$$

This definition is based on the definition of the least upper bound (*lub*) operation in the space $V_{\text{AT-MOST}}$: ignoring role-identity, we have $\leq$ on integers as the partial order, and *max* as the lub.[5] This of course is

[2] Note that the problem of actually computing `lcs` can be made less trivial by having Bit_n contain only some sufficiently large random subset of the bit vectors with $\frac{n}{2}$ 1's, so that the subclass hierarchy must be traversed to find the common (primitive) ancestors.

[3] Notice that we use $\Longrightarrow$ to denote the partial order on descriptions, and `Subsumes?` to denote the algorithm that implements this partial order.

[4] Informally, this term was introduced in [Patel-Schneider, 1989].

[5] To be precise, when we say that constructor k has partial order $\leq_k{}'$ and lub $\sqcup_k'$ "ignoring role identity" we mean that the domain is $V_k' \times \mathcal{R}ole$, the actual partial order is

$$(x_1, r_1) \leq_k{}' (x_2, r_2)) = (\text{if } r_1 = r_2 \text{ then } x_1 \leq_k{}' x_2 \text{ else } \texttt{false})$$

and the actual lub is

$$(x_1, r_1) \sqcup_k (x_2, r_2) = (\text{if } r_1 = r_2 \text{ then } x_1 \sqcup_k' x_2 \text{ else } \texttt{THING})$$

where `THING` denotes the set of all individuals. Most of the common constructors that deal with roles have relatively simple lubs and partial orders if role identity is ignored.

no accident: in general, if $\leq_k$ is being used to define the partial order used for subsumption, then the corresponding lub operation (if it exists) will define the lcs. This observation provides a connection between structural subsumption and lcs computation.

Theorem 6 *Suppose D1 and D2 are terms in normal form, so that every constructor k has its arguments taken from a join/upper semi-lattice*[6] $(V_k, \leq_k, \sqcup_k)$*. If subsumption is computed structurally based on* $\leq_k$*, then* lcs*(D1,D2) is non-empty, unique, and is specified by*

$$\texttt{lcs}(k_1(\alpha), k_2(\beta)) = \begin{cases} k_1(\alpha \sqcup_{k_1} \beta) & \text{if } k_1 = k_2 \\ \texttt{THING} & \text{otherwise} \end{cases}$$

Further, if both Subsumes? *and the computation of each* $\sqcup_k$ *takes polynomial time, then the computation of* lcs*(D1,D2) also takes polynomial time.*

The power of this theorem is most apparent when one considers constructors that can be composed together. For example, consider CLASSIC's ALL constructor, which has the associated partial order (ignoring role-identity) of $\Longrightarrow$. From the theorem, this definition, and the fact that lcs is the lub operator for the $\Longrightarrow$ partial order, we can derive a recursive definition of lcs for the ALL constructor (see Table 1).[7] In general, given a recursive structural Subsumes? algorithm and definitions of lcs for the base cases one can immediately construct a corresponding recursive lcs algorithm.

Observe also that the theorem allows a concise specification of lcs for an arbitrary set of constructors; this is again illustrated by the table, which provides such a specification for all of the constructors in the CLASSIC DL except for SAME-AS (which we will discuss in the next section.) The table can also be readily extended as new constructors are added. For example, if we introduce a concept constructor RANGE to represent Pascal-like integer intervals, then V_{RANGE} is $\mathcal{N} \times \mathcal{N}$, $(n1, n2) \leq_{\text{RANGE}} (m1, m2)$ iff $(m1 \leq n1) \wedge (n2 \leq m2)$, and $(n1, n2) \sqcup_{\text{RANGE}} (m1, m2) = (min(n1, m1), max(n2, m2))$.

Computing lcs for Attribute Chain Equalities

Some DLs support a constructor for testing the equality of attribute chains; an example is CLASSIC's SAME-AS constructor. Attribute chain equalities are useful in representing relationships between parts and subparts of an object; to take an example from natural language processing, one might define a sentence to be "reflexive" if its subject is the same as the direct object of the verb phrase, i.e. if `(SAME-AS (subj) (vp direct-obj))`. In this section, we will consider computing lcs for the SAME-AS constructor.

The semantics of SAME-AS can be concisely stated by noting that it is an equality relationship (i.e., it is reflexive, symmetric, and transitive) such that the equality of $\vec{A}$ and $\vec{B}$ is preserved by appending identical suffixes.[8] Consider the sublanguage $\text{DL}_{\text{SAME-AS}}$, containing only conjunctions of SAME-AS terms. Each such description D will partition the space $\mathcal{A}ttr^*$ of all attribute sequences into equivalence classes; let us denote the partition induced by these equivalence classes as π(D). A description D1 subsumes D2 iff π(D1) is a refinement of π(D2) (i.e., all equivalences in D1 also hold in D2). The lcs operator must therefore generate a description of the coarsest partition that is a refinement of both π(D1) and π(D2).

Unfortunately, such partitions cannot be represented directly, as there may be infinitely many equivalence classes, each of which may be infinitely large. Ait-Kaci [Ait-Kaci, 1984] has described a finite representation for these partitions called a *ψ-type*. Following Ait-Kaci, we will represent a ψ-type (and hence a partition) as a rooted directed graph (V, E, v_0) in which each edge is labeled with an attribute; such a graph will be called here an *equality graph (EQG)*. A partition is represented by constructing an EQG such that attribute sequence $\vec{A}$ is the SAME-AS attribute sequence $\vec{B}$ iff there are two paths in the EQG with labels $\vec{A}$ and $\vec{B}$ that lead from the root to the same vertex, or if $\vec{A}$ and $\vec{B}$ are the result of adding identical suffixes to two shorter sequences $\vec{A'}$ and $\vec{B'}$ where (SAME-AS $\vec{A'}$ $\vec{B'}$).

Ait-Kaci [1984] shows that ψ-types form a lattice, and presents efficient algorithms for constructing ψ-types, testing the partial order ($\Longrightarrow$) and constructing the glb (AND) of two ψ-types. In the remainder of this section, we present additional algorithms for constructing a ψ-type from a conjunction of SAME-AS restrictions, and for computing the lub (lcs) of two ψ-types.

To construct an EQG that encodes a description D, first add (to an initially empty graph) enough vertices and edges to create distinct paths labeled $\vec{A}$ and $\vec{B}$ for each restriction (SAME-AS $\vec{A}$ $\vec{B}$) in D. Then, merge[9] the final vertices of each of these pairs of paths. Finally, make the resulting graph "determin-

[6] A join/upper semi-lattice is a poset $(V_k, \leq_k)$ with an associated operation $\alpha \sqcup_k \beta$ that returns a unique least upper bound for each pair of elements α and β.

[7] Where $\mathcal{I}nd$={individuals}, $\mathcal{F}un$={host language functions}, $\mathcal{A}tom$={atoms}, and $\mathcal{D}esc$={descriptions}; $2^{\mathcal{X}}$ denotes the power set (e.g., $2^{\mathcal{I}nd}$ is the set of all sets of individuals) and $\mathcal{X}^*$ denotes the set of sequences (of arbitrary length) of members of $\mathcal{X}$. A single sequence is denoted $\vec{X} = (x_1, \ldots, x_n)$. Role identity is ignored for AT-MOST, AT-LEAST, and FILLS.

[8] I.e., (SAME-AS $\vec{A}$ $\vec{B}$) iff (SAME-AS $\vec{A}\vec{S}$ $\vec{B}\vec{S}$). Of course, attribute sequences not mentioned in the SAME-AS condition are in different equivalence classes.

[9] To *merge* two edges, one replaces their destination vertices with a single new vertex.

Constructor k	Domain V_k	Partial Order $\leq_k$	Upper Bound $\sqcup_k$
ALL	$\mathcal{R}ole \times \mathcal{D}esc$	$\lambda((r_1, D_1), (r_2, D_2))$. if $(r_1 = r_2)$ $D_1 \Longrightarrow D_2$ else false	$\lambda((r_1, D_1), (r_2, D_2))$. if $(r_1 = r_2)$ $\mathtt{lcs}(D_1, D_2)$ else THING
AT-LEAST	$\mathcal{N} \times \mathcal{R}ole$	$\geq$	min
AT-MOST	$\mathcal{N} \times \mathcal{R}ole$	$\leq$	max
FILLS	$2^{\mathcal{I}nd} \times \mathcal{R}ole$	$\supseteq$	$\cap$
ONE-OF	$2^{\mathcal{I}nd}$	$\subseteq$	$\cup$
TEST	$2^{\mathcal{F}un}$	$\supseteq$	$\cap$
PRIM	$\mathcal{A}tom$	$=$	$\lambda(p_1, p_2)$. if $(p_1 = p_2)$ p_1 else THING
AND	$\mathcal{D}esc^*$	$\lambda(\vec{C}, \vec{D})$. $\wedge_i \vee_j (C_j \Longrightarrow D_i)$	$\lambda(\vec{C}, \vec{D})$. $\mathrm{AND}_{i,j}\mathtt{lcs}(C_j, D_i)$

Table 1: Structural subsumption and lcs rules for common constructors.

istic" by repeatedly merging edges that exit from the same vertex and have the same label.[10] When this process is finished, every vertex v in the constructed EQG will represent an equivalence class of attribute sequences—namely those sequences that label paths to v from the root of the EQG. (See Figure 1 for examples.) LCS computation can be done by performing an analog of the "cross-product" construction for finite-automaton intersection [Hopcroft and Ullman, 1979] — i.e., one forms a new graph with vertex set $V_1 \times V_2$, root (v_{01}, v_{02}), and edge set $\{((v_1, v_2), (w_1, w_2), a) : (v_1, w_1, a) \in E_1$ and $(v_2, w_2, a) \in E_2\}$; here a tuple (v, w, a) denotes an edge from v to w with label a. This construction can be performed in polynomial time, and yields an EQG of size no greater than product of the sizes of the two input EQG's.

By the above comments, $\mathrm{DL}_{\mathrm{SAME\text{-}AS}}$ is a DL for which subsumption and lcs are tractable. However, although by Table 1 CLASSIC *without* SAME-AS is also tractable in the same sense, the SAME-AS and ALL constructors interact in subtle ways, making a straightforward integration of the two DLs impossible (more precisely, the interaction makes it impossible to normalize in a manner that preserves the semantics for ALL presented in Table 1.) The technique of this section can, however, be extended to the full CLASSIC language, as is detailed in [Cohen *et al.*, 1992].[11]

Computing lcs for Sets of Descriptions

In some applications, notably inductive learning, it is necessary to find the commonalities between relatively large sets of objects. For this reason, we will now consider the problem of computing the lcs of a set of objects, rather than merely a pair. In the following discussion, $|D|$ denotes the size of a description D.

Observe that by definition lcs is commutative and associative: thus it is always possible to compute $\mathtt{lcs}(D_1, \ldots, D_n)$ by a series of pairwise lcs computations. However, problems may arise even if the lcs of a pair of descriptions is tractably computable:

Theorem 7 *There exists a a polynomial function $p(n)$ so that for all n $\exists D_1, \ldots, D_n \in \mathrm{DL}_{\mathrm{SAME\text{-}AS}}$ such that $|D_i| = O(n)$ for each D_i, but $|\mathtt{lcs}(D_1, ..., D_n)| = \Omega(\frac{2^n}{p(n)})$.*

Proof: Consider the descriptions $D_1, D_2, ..., D_n$, with D_i defined as

$$D_i = (\mathrm{AND}\ \ (\mathrm{AND}_{j\neq i}\ (\mathrm{SAME\text{-}AS}\ ()\ (a_j))) \\ (\mathrm{AND}_{j\neq i}\ (\mathrm{SAME\text{-}AS}\ (a_i)\ (a_i\ a_j))) \\ (\mathrm{SAME\text{-}AS}\ ()\ (a_i\ a_i)))$$

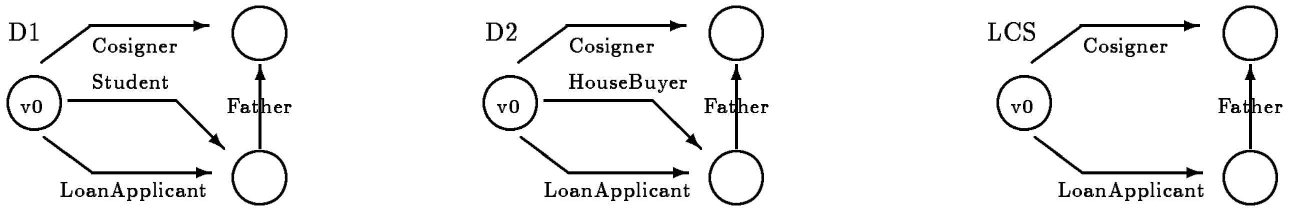
where the a_i's are all distinct attributes. The partition $\pi(D_i)$ has two equivalence classes: two attribute sequences are equal iff they contain the same number of occurrences (modulo 2) of attribute a_i. Recall that for $D_{lcs} = \mathtt{lcs}(D_1, \ldots, D_n)$, $\pi(D_{lcs})$ will be the coarsest partition that is a refinement of each $\pi(D_i)$; this means that two attribute sequences will be considered equal by D_{lcs} iff they are considered equal by all of the D_i's, i.e., if they contain the same number of occurrences (modulo 2) of all n attributes. Thus $\pi(D_{lcs})$ contains at least 2^n equivalence classes, and hence its equality graph must contain at least 2^n nodes. Finally, since converting a description D to an EQG requires time only polynomial in $|D|$, no equivalent description D can be more than polynomially smaller. ■

In the theorem, exponential growth occurs because $|\mathtt{lcs}(D_1, D_2)|$ is bounded only by $|D_1| \cdot |D_2|$; hence $\mathtt{lcs}(D_1, \ldots, D_n)$ can be large. We note that exponential behavior cannot occur without this multiplicative growth:

Proposition 8 *Under the conditions of Theorem 6, if $|\alpha \sqcup_k \beta| \leq |\alpha| + |\beta|$, then $\mathtt{lcs}(D_1, \ldots, D_n)$ can be computed in polynomial time.*

[10] Making the graph deterministic makes it possible to tractably test subsumption, as one can tractably check if one deterministic graph is a subgraph of another.

[11] Briefly, one can distribute information about the other constructors through an EQG, tagging the vertices of the EQG with ONE-OF and PRIM constraints and the edges of the EQG with role-related restrictions; efficient (but non-structural) Subsumes?, AND, and lcs algorithms can then be defined on this data structure.




```
(AND (SAME-AS (LoanApplicant)
              (Student))
     (SAME-AS (Cosigner)
              (Student Father)))

(AND (SAME-AS (LoanApplicant)
              (HouseBuyer))
     (SAME-AS (Cosigner)
              (HouseBuyer Father)))

(AND
 (SAME-AS
  (Cosigner)
  (LoanApplicant Father)))
```

Figure 1: Two EQG's and their `lcs`

From Table 1, we see that for CLASSIC without SAME-AS, `lcs` is tractable in this strong sense.

Conclusions

This paper has considered a new operation for description logics: computing the *least common subsumer (LCS)* of a pair of descriptions. This operation is a way of computing the largest set of commonalities between two descriptions, and has applications in areas including inductive learning and knowledge-based vivification. We have analyzed the LCS operation, primarily by analyzing its relationship to the well-studied and well-understood problem of subsumption. First, we showed that no close relationship holds in the general case. We then defined the class of *structural subsumption relations* (which apply to a large portion of extant DLs) and showed that, in this restricted setting, the relationship between LCS and subsumption is quite close; making use of this relationship, we also developed a very general and modular implementation of LCS. Finally, we presented a method for computing the LCS of attribute chain equalities, and presented some additional results on the tractability of computing the LCS of a set of descriptions; the latter problem is important in learning.

We have seen that constructors such as SAME-AS whose subsumption is not easily described in a "structural" manner can cause complications in the computation of LCS. For this reason, we have postponed consideration of generalizations of SAME-AS which are known to make subsumption undecideable, such as SUBSET-OF and SAME-AS applied to role (as opposed to attribute) chains. We also leave as further work analysis of LCS for DLs allowing recursive concept definitions, role hierarchies, or role constructors such as inversion and transitive closure.

Acknowledgements

The authors would like to thank Ron Brachman, Peter Patel-Schneider, and Bart Selman for comments on a draft of the paper; we are also grateful to Tony Bonner and Michael Kifer for pointing out a technical error in an earlier version of the paper. Other colleagues at Bell Labs contributed through many helpful discussions.

Appendix: The CLASSIC 1.0 DL

The following are the description constructors for CLASSIC, and their syntax.

(AT-LEAST n r) : the set of individuals with at least n fillers of role r.

(AT-MOST n r) : the set of individuals with at most n fillers of role r.

(ALL r D) : the set of individuals for which all of the fillers of role r satisfy D.

(AND $D_1 \ldots D_n$) : the set of individuals that satisfy all of the descriptions $D_1, \ldots, D_n$.

(FILLS r $I_1 \ldots I_n$) : the set of individuals for which individuals $I_1, \ldots, I_n$ fill role r.

(ONE-OF $I_1 \ldots I_n$) : denotes either the set $\{I_1\}$ or $\ldots$ or $\{I_n\}$.

(TEST $T_1 \ldots T_n$) : tests (arbitrary host language predicates) $T_1, \ldots, T_n$ are true of the instances of the concept. This is essentially an escape-hatch to the host language, which enables a user to encode procedural sufficiency conditions.

(PRIM i) : denotes primitive concept i. A primitive concept is a name given to a "natural kind"—a set of objects for which no sufficient definition exists.

(SAME-AS $(r_{1,1} \ldots r_{1,k})$ $(r_{2,1} \ldots r_{2,k})$) : The set of individuals for which the result of following the first chain of attributes is the same as the result of following the second chain of attributes.

References

(Ait-Kaci, 1984) Hassan Ait-Kaci. A lattice theoretic approach to computation based on a calculus of partially ordered type structures. PhD Thesis, University of Pennsylvania, 1984.

(Beck *et al.*, 1989) H. Beck, H. Gala, and S. Navathe. Classification as a query processing technique in the CANDIDE semantic model. In *Proceedings of the Data Engineering Conference*, pages 572–581, Los Angeles, California, 1989.

(Bergamaschi *et al.*, 1988) S. Bergamaschi, F.Bonfatti, and C. Sartori. Entity-situation: a model for the knowledge representation module of a kbms. In *Advances in Database Technology: EDBT'88*. Springer-Verlag, 1988.

(Borgida and Etherington, 1989) A. Borgida and D. Etherington. Hierarchical knowledge bases and efficient disjunctive reasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ontario, 1989.

(Borgida *et al.*, 1989) A. Borgida, R. J. Brachman, D. L. McGuinness, and L. Resnick. CLASSIC: A structural data model for objects. In *Proceedings of SIGMOD-89*, Portland, Oregon, 1989.

(Brachman *et al.*, 1983) R. J. Brachman, R. E. Fikes, and H. J. Levesque. Krypton: A functional approach to knowledge representation. *IEEE Computer*, 16(10):67–73, 1983.

(Buntine, 1988) Wray Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36(2):149–176, 1988.

(Cohen and Hirsh, 1992) W. Cohen and H. Hirsh. Learnability of description logics. In preparation, 1992.

(Cohen *et al.*, 1992) W. Cohen, H. Hirsh, and A. Borgida. Learning in description logics using least common subsumers. In preparation, 1992.

(Devanbu *et al.*, 1991) P. Devanbu, R. J. Brachman, P. Selfridge, and B. Ballard. LaSSIE: A knowledge-based software information system. *Communications of the ACM*, 35(5), May 1991.

(Edelman and Owsnicki, 1986) J. Edelman and B. Owsnicki. Data models in knowledge representation systems. In *Proceedings of GWAI-86*. Springer-Verlag, 1986.

(Flann and Dietterich, 1989) Nicholas Flann and Thomas Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4(2), 1989.

(Frisch and Page, 1990) A. Frisch and C. D. Page. Generalization with taxonomic information. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, Massachusetts, 1990. MIT Press.

(Hirsh, 1992) Haym Hirsh. Polynomial-time learning with version spaces. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, California, 1992. MIT Press.

(Hopcroft and Ullman, 1979) John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

(Levesque, 1986) Hector Levesque. Making believers out of computers. *Artificial Intelligence*, 30:81–108, 1986. (Originally given as the "Computers and Thought" lecture at IJCAI-85.).

(Mays *et al.*, 1987) E. Mays, C. Apte, J. Griesmer, and J. Kastner. Organizing knowledge in a complex financial domain. *IEEE Expert*, pages 61–70, Fall 1987.

(Patel-Schneider, 1984) P. F. Patel-Schneider. Small can be beautiful in knowledge representation. In *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems*, pages 11–16, Denver, Colorado, 1984.

(Patel-Schneider, 1989) P. F. Patel-Schneider. A four-valued semantics for terminological logics. *Artificial Intelligence*, 38:319–351, 1989.

(Plotkin, 1969) G. D. Plotkin. A note on inductive generalization. In *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, 1969.

(Schewe, 1989) Klaus Dieter Schewe. Variant construction using constraint propagation techniques over semantic networks. In *Proceedings of the 5th Austrian AI Conference*, Insbruck, 1989.

(Valiant, 1984) L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11), November 1984.

(Woods and Schmolze, 1992) W. A. Woods and J. G. Schmolze. The KL-ONE family. *Computers And Mathematics With Applications*, 23(2-5), March 1992.