

A Tableaux-Based Method for Computing Least Common Subsumers for Expressive Description Logics

Francesco M. Donini
Università della Tuscia
Viterbo, Italy
donini@unitus.it

Simona Colucci
Politecnico di Bari
Italy
s.colucci@poliba.it

Tommaso Di Noia
Politecnico di Bari
Italy
t.dinoia@poliba.it

Eugenio Di Sciascio
Politecnico di Bari
Italy
disciascio@poliba.it

Abstract

Least Common Subsumers (LCS) have been proposed in Description Logics (DL) to capture the commonalities between two or more concepts. Since its introduction in 1992, LCS have been successfully employed as a logical tool for a variety of applications, spanning from inductive learning, to bottom-up construction of knowledge bases, information retrieval, to name a few. The best known algorithm for computing LCS uses structural comparison on normal forms, and the most expressive DL it is applied to is $\mathcal{AL}\mathcal{EN}$. We provide a general tableau-based calculus for computing LCS, via substitutions on concept terms containing concept variables. We show the applicability of our method to an expressive DL (but without disjunction and full negation), discuss complexity issues, and show the generality of our proposal.

1 Motivation

Least Common Subsumers (LCSs) in Description Logics (DLs) have been introduced by Cohen *et al.*[1992] to denote the most specific concept descriptions subsuming all of the elements of a given collection of concepts.

Since its introduction, LCS has been usefully exploited in several application fields where the search for commonalities in a collection is needed, despite the complexity of its computation even for inexpressive DLs.

As motivation for our work stems from an actual need, we start recalling usefulness of LCS as witnessed by previous works. One of the best known applications of LCS is the support to ontology design ([Baader and Küsters, 1998],[Kopena and Regli, 2003], [Ovchinnikova *et al.*, 2007]). In particular, Lutz *et al.* [2006] recently underlined the need to have LCS available also for the more expressive languages currently used in ontology design.

LCS has been also widely used in semantic-based information retrieval, for the definition of measures for concept similarity ([Möller *et al.*, 1998], [Janowicz *et al.*, 2008]).

Hacid *et al.* [2002] studied using LCS for defining and computing the best covering problem. Such a computation approach have been applied to semantic Web services discovery and composition [Benatallah *et al.*, 2005] and, more

recently, to the composition of learning resources [Karam *et al.*, 2007]. Hacid *et al.* [2000] also employed LCS for schema extraction from semistructured data.

A further well-known application field is in inductive learning algorithms[Cohen and Hirsh, 1994].

Colucci *et al.* [2008] addressed the knowledge management problem of evaluating the Core Competence of a company, introducing inferences computed by exploiting LCS.

The variety of approaches recalled so far witnesses the importance of LCS computation in knowledge representation and reasoning literature, justifies the need for LCS in expressive DLs, and motivates our contribution, which introduces a novel general tableau-based calculus for computing LCS. The approach uses substitutions on concept terms containing concept variables. We present our method with reference to a DL more expressive than $\mathcal{AL}\mathcal{EN}$, namely $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$.

The rest of the paper is organized as follows: in the next section previous computation results for LCS in different DLs are recalled, before introducing the DL used in the paper in Section 3. We then present a novel generalized definition of LCS in Section 4. Based on such definition and on the tableau rules for $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$ presented in Section 5, we propose a tableaux-based method for LCS computation in Section 6. Conclusions and future research directions close the paper.

2 Previous computation results

LCS computation has been investigated for a limited number of DLs, due to the complexity of the related problem.

Baader *et al.*[1999] propose to compute LCS as graph product of description trees representing \mathcal{EL} , $\mathcal{FL}\mathcal{E}$ and $\mathcal{AL}\mathcal{E}$ concept descriptions.

An approach for computing LCS of cyclic $\mathcal{AL}\mathcal{N}$ concept descriptions has been also proposed by Baader and Küsters, which exploit automata-theoretic characterizations of value-restriction sets [1998].

Brandt *et al.* investigated on the problem of computing LCS in the presence of DLs with transitive roles [2003]. The authors present an algorithm based on structural comparison of normal forms for computing LCS of \mathcal{FL}_0^+ concept descriptions, and extend the algorithms by Baader *et al.* [1999] to \mathcal{EL}^+ , $\mathcal{EL}\mathcal{H}^+$ and $\mathcal{FL}\mathcal{E}^+$. The paper does not deal with complexity results in the presence of transitive roles, but it is well-known that the complexity for LCS computation arises

to exponential size even for DLs with only existential restriction, namely \mathcal{EL} [Baader *et al.*, 1999] and can neither be reduced by introducing TBoxes to shorten possible repetitions [Baader and Turhan, 2002].

So far, the most expressive investigated DL is $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}$, for which a double exponential time algorithm, based on structural comparison of normal forms, has been proposed [Küstners and Molitor, 2005], and no proposal is available for $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$.

3 The Language $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$

To show the generality of our calculus, we choose the DL $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$, which is $\mathcal{SH}\mathcal{I}\mathcal{N}$ without constructs that introduce concept disjunction, namely, \sqcup and \neg . In languages including disjunction, the simplest LCS would be just $C_1 \sqcup C_2$ —or equivalently, $\neg(\neg C_1 \sqcap \neg C_2)$ with full negation—making the LCS problem trivial.

Let N_r be a set of role names. A general role R can be either a role name $P \in N_r$, or its inverse P^- . We admit a set of *role axioms*, formed by: (1) a *role hierarchy* \mathcal{H} , which is a set of role inclusions of the form $R_1 \sqsubseteq R_2$, and (2) a set of transitivity axioms for roles, denoted by $\text{trans}(R)$. We denote by \sqsubseteq^* the transitive closure of $\mathcal{H} \cup \{R^- \sqsubseteq S^- \mid S \sqsubseteq R \in \mathcal{H}\}$. A role S is *simple* if it is not transitive, and for no R such that $R \sqsubseteq^* S$, R is transitive.

In the following syntax for concepts, let A be any concept name in a set N_c of concept names, let R be a role, and S be a simple role.

$$\begin{aligned} At &\longrightarrow \top \mid \perp \mid A \mid \neg A \mid (\geq n S) \mid (\leq n S) & (1) \\ C &\longrightarrow At \mid C_1 \sqcap C_2 \mid \exists R.C_1 \mid \forall R.C_1 & (2) \end{aligned}$$

We call At an *atomic concept*, while C simply *concept*. We consider $(\leq R0)$ as an abbreviation of $\forall R.\perp$. We extend the inverse role constructor to general roles by letting $R^- = P^-$ if $R = P$, and $R^- = P$ if $R = P^-$. Moreover, we denote by $\sim C$ the *negation normal form* of $\neg C$ (see Baader *et al.* [2003] (Ch.2) for a definition).

As every DL, $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$ is equipped with a model-theoretic semantics. We regret to skip a detailed definition for lack of space, referring the interested reader to Baader *et al.* [2003, Ch.2]. We denote by $C \sqsubseteq D$ subsumption between two concepts C and D . *Strict* subsumption is denoted by $C \sqsubset D$, meaning both $C \sqsubseteq D$ and $D \not\sqsubseteq C$. Intuitively, subsumption is interpreted as subset inclusion between the sets interpreting C and D ; strict subsumption is strict subset inclusion. In the rest of the paper, we use concepts C_1, C_2, D and L that always belong to $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$. Moreover, we make use of the following property.

Proposition 1 *For every four concepts C_1, C_2, D_1 , and D_2 : if $C_1 \sqsubseteq D_1$ and $C_2 \sqsubseteq D_2$, then $C_1 \sqcap C_2 \sqsubseteq D_1 \sqcap D_2$.*

As for concept axioms, we do not consider General Concept Inclusions (GCI) in this paper, since Baader *et al.* [2007] showed that even for the simple DL $\mathcal{AL}\mathcal{E}$ LCS may not exist when GCIs interpreted with descriptive semantics are used. We could admit *simple* concept inclusions—*e.g.*, acyclic definitions of concept names—but since they do not add expressivity to the language, for sake of simplicity we do not consider them.

Regarding Qualified Number Restrictions (\mathcal{Q}), observe that $(\leq 0 P.(\sim C_1 \sqcap \sim C_2)) \equiv \forall P.(C_1 \sqcup C_2)$. Hence, qualified at-most number restrictions would implicitly introduce \sqcup in concepts—although inside quantifications—so we want to exclude them. Since in DLs the at-most restriction comes paired with the at-least restriction, we exclude them both, only for uniformity.

4 A logical definition of the LCS

In this section we denote by \mathcal{DL} a generic Description Logic.

Definition 1 (LCS as a set) *Let $C_1, C_2 \in \mathcal{DL}$ be two concepts. By $LCS(C_1, C_2)$ we mean a set of equivalent concepts in \mathcal{DL} , such that for every $L \in LCS(C_1, C_2)$, (a) L is a common subsumer of C_1 and C_2 —in formulas, $C_1 \sqsubseteq L, C_2 \sqsubseteq L$ —and (b) there does not exist $D \in \mathcal{DL}$ such that $C_1 \sqsubseteq D, C_2 \sqsubseteq D$ and $D \sqsubset L$.*

In order to write a second-order formula defining LCS, we need an alphabet $N_x = \{X_0, X_1, X_2, \dots\}$ of concept variables, which we can quantify over.

Theorem 1 *$L \in LCS(C_1, C_2)$ iff $C_1 \sqsubseteq L, C_2 \sqsubseteq L$, and the following formula is false:*

$$\exists X. \{X \in \mathcal{DL}, C_1 \sqsubseteq X, C_2 \sqsubseteq X, X \sqsubset L\} \quad (3)$$

where commas in (3) abbreviate conjunction. The proof is straightforward, since (3) is just a formal rewriting of (b) in Def.1. Observe that (3) does not belong to the Monadic Second-Order fragment proved decidable by Rabin [1969], since the formula $X \in \mathcal{DL}$, if explicitly defined, would force to write a least fixpoint to logically define \mathcal{DL} . We prefer instead to keep $X \in \mathcal{DL}$ as a constraint on the possible assignments for X , which restricts the possible substitutions for X to be defined later on. Intuitively, we are using *general semantics* for interpreting X [Henkin, 1950].

To introduce a more computation-oriented version of (3), we define the *decoration* of a concept. Intuitively, given a concept L , we put a new concept variable in conjunction with the filler of every universal and existential role quantification in L , plus one concept variable in the outermost level of L . Since we would like to have all variables consecutively numbered, starting at 0, we define the decoration in Algorithm 1 by means of a recursive procedure, plus a global counter i that keeps track of the last index used in whatever recursive call. Although maybe not mathematically elegant, we believe that Algorithm 1 presents this idea in the most intuitive way.

Definition 2 *Let C be a concept in \mathcal{DL} , and N_x be an alphabet of concept variables. We denote by C_X the decoration of C , defined by Algorithm 1.*

For example, if C is the concept $A \sqcap \exists P.(\geq 2 Q) \sqcap \forall P.\forall Q.B$, then $C_X = X_0 \sqcap \exists P.(X_1 \sqcap (\geq 2 Q)) \sqcap \forall P.(X_2 \sqcap \forall Q.(X_3 \sqcap B))$. Note that C_X is a (particular) *concept term* [Baader and Narendran, 2001], *i.e.*, a concept formed according to the rules in (1) and (2), with the addition to (1) of the rule $At \longrightarrow X$, for every X in N_x . Our decorations are particular concept terms, in that every variable occurs only once in the term, and—by inspecting Algorithm 1—one can verify that variables are one-one with quantifications, plus the outermost

Algorithm 1 (Decoration of a concept C)

input concept $C \in \mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$;
var $i := 0$;
return $C_X \doteq X_0 \sqcap \text{dec}(C)$

function $\text{dec}(C)$

case C is of the form

atomic: **return** C

$C_1 \sqcap C_2$: **return** $\text{dec}(C_1) \sqcap \text{dec}(C_2)$

$\exists R.C_1$: $i := i+1$; **return** $\exists R.(X_i \sqcap \text{dec}(C_1))$

$\forall R.C_1$: $i := i+1$; **return** $\forall R.(X_i \sqcap \text{dec}(C_1))$

end function

variable. Observe that we do *not* count number restrictions as quantifiers, although their logical definition would contain a quantifier.

Definition 3 (Substitutions) A substitution σ is a set of pairs $\{X_{i_1} \mapsto D_{i_1}, \dots, X_{i_k} \mapsto D_{i_k}\}$, where indexes i_1, \dots, i_k are all different, and for every $j = 1, \dots, k$ each X_{i_j} is a concept variable and each D_{i_j} is a concept term. A substitution is ground if every D_{i_j} contains no variables, i.e., $D_{i_j} \in \mathcal{DL}$.

For a decorated concept C_X , we inductively define $\sigma(C_X)$ as $\sigma(X_i) = D_i$, $\sigma(\neg X_i) = \sim D_i$, $\sigma(C) = C$ if C is atomic, $\sigma(C_1 \sqcap C_2) = \sigma(C_1) \sqcap \sigma(C_2)$, $\sigma(\exists R.C) = \exists R.\sigma(C)$, $\sigma(\forall R.C) = \forall R.\sigma(C)$.

The cardinality of a substitution is its cardinality as a set.

Let $\sigma_{\top, n}$ denote the substitution $\{X_i \mapsto \top\}_{i=0, \dots, n}$; since variables in decorations appear always in some conjunction, then for every concept $C \in \mathcal{DL}$ containing n quantifications, $\sigma_{\top, n}(C_X) \equiv C$.

We also need *renaming* of variables, as bijective functions on the indices of variables. We denote a renaming by ρ . A variable renaming can also be applied to a substitution, $\rho(\sigma)$, meaning that the index of each variable changes according to ρ ; e.g., if $\sigma = \{X_0 \mapsto A, X_1 \mapsto B\}$, and ρ is the renaming $\{0 \mapsto 1, 1 \mapsto 2\}$, then $\rho(\sigma) = \{X_1 \mapsto A, X_2 \mapsto B\}$.

We can now reformulate Thm.1 (for $\mathcal{DL} = \mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$) in a way that leads to a direct computational method.

Theorem 2 Let $C_1, C_2, L \in \mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$. Then $L \in \text{LCS}(C_1, C_2)$ iff $C_1 \sqsubseteq L, C_2 \sqsubseteq L$, and the formula below is false:

$$\exists \sigma \{ \sigma \text{ is ground, } C_1 \sqsubseteq \sigma(L_X), C_2 \sqsubseteq \sigma(L_X), L \not\sqsubseteq \sigma(L_X) \} \quad (4)$$

Proof. Since Thm.1 has the same premises of Thm.2, the latter amounts to an equivalence between formulas (3) and (4), which we prove below. Let $0, \dots, n$ be the indexes of concept variables in L_X .

(3) \Rightarrow (4) If (3) holds, then there is a concept $L_0 \in \mathcal{DL}$ such that all three conditions: $C_1 \sqsubseteq L_0, C_2 \sqsubseteq L_0$, and $L_0 \sqsubseteq L$ hold. Then let σ be the substitution $\{X_0 \mapsto L_0\} \cup \{X_i \mapsto \top\}_{i=1, \dots, n}$. Applying σ to L_X we obtain $L_0 \sqcap \sigma_{\top, n}(\text{dec}(L))$, which is equivalent to L_0 since $\sigma_{\top, n}(\text{dec}(L)) \equiv L$ and $L_0 \sqsubseteq L$. Since all conditions for $\sigma(L_X)$ hold, the claim follows.

(4) \Rightarrow (3) If (4) holds, then we use $\sigma(L_X)$ as a witness for $\exists X$ in (3), proving all four conditions of (3). First condition is met because $\sigma(L_X) \in \mathcal{DL}$ since σ is ground. Second and third

conditions hold by hypothesis. Regarding fourth condition (proving $\sigma(L_X) \sqsubseteq L$), we prove subsumption separately (to ease readability) in Lemma 1 below, while fourth condition in (4) implies that subsumption holds strictly. \square

Lemma 1 For every given concept $C \in \mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$, and every ground substitution σ , it holds $\sigma(C_X) \sqsubseteq C$.

Proof. By induction on the structure of C . Base cases: when C is atomic (no structure), $C_X = X_0 \sqcap C$ contains just one concept variable. Then $\sigma(X_0 \sqcap C) = \sigma(X_0) \sqcap C = D_0 \sqcap C \sqsubseteq C$ by definition of \sqcap , whatever D_0 , and hence for every σ .

Inductive cases: we now show that the claim holds for $\exists R.C_1$, given that it holds for C_1 which is structurally simpler. Suppose C_1 has $n \geq 0$ quantifiers, hence $(C_1)_X$ has $n+1$ variables X_0, \dots, X_n . Observe that by letting $\rho = \{i \mapsto i+1\}$, it holds $(\exists R.C_1)_X = X_0 \sqcap \exists R.\rho((C_1)_X)$, where the renaming is necessary because in $(C_1)_X$ variables are numbered from 0. Observe that ρ is a bijection, hence its inverse ρ^{-1} is well defined. Then, for every substitution σ over $n+2$ variables, $\sigma((\exists R.C_1)_X) = \sigma(X_0) \sqcap \exists R.\sigma(\rho((C_1)_X))$. Let σ' be σ without $X_0 \mapsto D_0$, and let $\sigma_1 = \rho^{-1}(\sigma')$. Then $\sigma(\rho((C_1)_X)) = \sigma_1((C_1)_X)$, which is subsumed by C_1 by inductive hypothesis. Hence $\exists R.\sigma(\rho((C_1)_X)) \sqsubseteq \exists R.C_1$, and the subsumption holds also if (whatever concept) D_0 is conjoined on the left-hand side. Since we made no restrictions on σ , for every σ , the claim $\sigma((\exists R.C_1)_X) \sqsubseteq \exists R.C_1$ holds. A similar proof can be laid out for $\forall R.C_1$.

Regarding $C_1 \sqcap C_2$, the inductive hypothesis is that the claim holds for C_1 and C_2 separately, which are both structurally simpler. Again, suppose that both C_1 and C_2 have at most $m, n \geq 0$ quantifiers, respectively, and let ρ be now the renaming $\{i \mapsto i+m\}_{i \geq 1}$. For this renaming, $(C_1 \sqcap C_2)_X \equiv (C_1)_X \sqcap \rho((C_2)_X)$, where *both* decorations introduce a variable X_0 (which ρ does not rename, since it applies to $i \geq 1$), and equivalence holds since $X_0 \sqcap X_0 \equiv X_0$. Then, for every σ on $m+n+1$ variables, $\sigma((C_1 \sqcap C_2)_X) \equiv \sigma_1((C_1)_X) \sqcap \sigma_2((C_2)_X)$, where σ_1 is just σ restricted to the first $m+1$ variables, while σ_2 is built as follows: let σ' be σ restricted to the last n variables; let $\rho' = \{i+m \mapsto i\}_{i=m+1, \dots, n}$. Then, $\sigma_2 = \{X_0 \mapsto D_0\} \cup \rho'(\sigma')$. By inductive hypothesis, both $\sigma_1((C_1)_X) \sqsubseteq C_1$ and $\sigma_2((C_2)_X) \sqsubseteq C_2$ hold. By Prop.1, the claim is obtained. \square

Observe that Thm. 2 refers to $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$ only because both Algorithm 1 and Lemma 1 refer to $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$.

5 Tableaux Rules for $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$

We first give an intuition of the way our calculus proceeds. To prove or disprove Formula (4), we expand three tableaux, one for each of the three conditions: \mathbf{T}_1 for $C_1 \sqsubseteq L_X$, \mathbf{T}_2 for $C_2 \sqsubseteq L_X$, and \mathbf{T}_3 for $L \not\sqsubseteq L_X$. The tableaux are first expanded using tableaux rules (T-rules), treating concept variables as concept names. Then, by using substitution rules (S-rules), we try to find a substitution σ satisfying (4), i.e., closing \mathbf{T}_1 and \mathbf{T}_2 and leaving \mathbf{T}_3 open. The substitution might make applicable some other T-rule, and so on, till no rule is applicable. If all branches of \mathbf{T}_1 and \mathbf{T}_2 close, and at least one branch of \mathbf{T}_3 is open, we found a substitution σ

<p>All rules are applicable only if x is <i>not blocked</i>. For each $i = 1, 2, 3$, \mathcal{L}_i is a branch in \mathbf{T}_i.</p> <p>\sqcap-rule : if $C \sqcap D \in \mathcal{L}_i(x)$, then add both C and D to $\mathcal{L}_i(x)$</p> <p>\sqcup-rule : if $C \sqcup D \in \mathcal{L}_i(x)$, then add either C or D to $\mathcal{L}_i(x)$</p> <p>\exists-rule : if $\exists R.C \in \mathcal{L}_i(x)$, and x has no R-successor y with $C \in \mathcal{L}_i(y)$, then pick up a new individual y, add R to $\mathcal{L}(x, y)$, and let $\mathcal{L}_i(y) := \{C\}$</p> <p>$\forall$-rule : if $\forall R.C \in \mathcal{L}_i(x)$, and there exists an individual y such that y is an R-successor of x, then add C to $\mathcal{L}_i(y)$.</p> <p>\forall_+-rule : if $\forall S.C \in \mathcal{L}_i(x)$, with $\text{trans}(R)$ and $R \sqsubseteq^* S$, there exists an individual y such that y is an R-successor of x, and $\forall R.C \notin \mathcal{L}_i(y)$, then add $\forall R.C$ to $\mathcal{L}_i(y)$</p> <p>\geq-rule : if $(\geq n S) \in \mathcal{L}_i(x)$, and x has not n S-neighbors y_1, \dots, y_n with $y_\ell \neq y_j$ for $1 \leq \ell < j \leq n$, then create n new successors y_1, \dots, y_n of x with $\mathcal{L}_i(x, y_\ell) = \{S\}$, and $y_\ell \neq y_j$, for $1 \leq \ell < j \leq n$</p> <p>\leq-rule : if $(\leq n S) \in \mathcal{L}_i(x)$ with $n \geq 1$, and there are more than n S-neighbors of x, and there are two S-neighbours y, z of x, y is an S-successor of x, and not $y \neq z$ then (1) add $\mathcal{L}_i(y)$ to $\mathcal{L}_i(z)$, (2) for every $R \in \mathcal{L}_i(x, y)$ if z is a predecessor of x then add R^- to $\mathcal{L}_i(z, x)$ else add R to $\mathcal{L}_i(x, z)$, (3) let $\mathcal{L}_i(x, y) = \emptyset$, and (4) for all u with $u \neq y$, set $u \neq z$</p>
--

Figure 1: Tableaux rules (T-rules) for $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$ (rephrased from Tobies [2001, p.128])

validating (4), otherwise, we prove that no such σ exist, disproving (4). We warn the reader that the calculus we present in this section does *not* compute an LCS; it just tries to prove Formula (4), by exhibiting a common subsumer of C_1, C_2 which is “better” than L . In the next section, we use such a calculus to compute—in an incremental fashion—a finite LCS (if one exists).

Rules for constructing tableaux in $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$ (Fig. 1) are a subset of the ones for \mathcal{SHIQ} , and have been proved [Tobies, 2001] sound and complete. We summarize them here for sake of completeness, remarking that we just inherit them from past research. Any inaccuracy is due to our rephrasing. In such rules, blocking is *pair-wise* blocking as defined by, e.g., Tobies [2001] (p.125); our only addition is that concept variables are treated as concept names for what regards blocking.

We recall that an individual y is an S -successor of x in \mathcal{L}_i , (for $i = 1, 2, 3$) if for some role R , both $R \in \mathcal{L}_i(x, y)$ and $R \sqsubseteq^* S$. Conversely, y is an S -predecessor of x if x is an S -successor of y . An individual y is an R -neighbor of x if either y is an R -successor of x , or x is an R^- -successor of y . The definitions of successor, predecessor, and neighbor allow us to treat roles and inverse roles in a uniform way, both in T-rules (Fig. 1) and in subsequent S-rules (Fig. 2).

Differently from Tobies [2001], we say that T-rules construct a *branch* \mathcal{L} , while we call *tableau* the set of all different branches that can be constructed applying T-rules. Branches are different because of the nondeterminism present in \sqcup -rule and \leq -rule (we ignore differences due to possible renaming of new individuals in \geq -rule). A branch \mathcal{L} is *closed* if for some individual x , either (a) $\perp \in \mathcal{L}(x)$, or (b) $\{A, \neg A\} \subseteq$

$\mathcal{L}(x)$ for some $A \in \mathbf{N}_c$, or (c) $(\leq n R) \in \mathcal{L}(x)$ and there are $n + 1$ R -neighbors y_1, \dots, y_{n+1} of x such that $y_i \neq y_j \in \mathcal{L}$ for $1 \leq i < j \leq n + 1$. A branch is *T-complete* if no T-rule is applicable. A tableau is *closed* if all its T-complete branches are closed, it is *open* if there exists at least one T-complete branch which is not closed.

We now give a sketchy intuition about T-rules, referring the interested reader to Baader *et al.* [2003, Ch.2]. A tableau for $C \in \mathcal{L}(a)$ tries to construct a model for this formula. A T-complete, open branch identifies such a model, in which individuals that are pairwise blocked (say, b) represent potentially infinitely many individuals all with the same properties (“copies” of b). A closed branch, instead, exhibits a plain contradiction, hence no model can be defined from it; so when all branches are closed, the initial formula is proved to be unsatisfiable.

We are now able to present our original part of this section, namely, *substitution* rules (S-rules) dealing with concept variables, in Fig. 2. There is a substitution rule for every syntax rule in (1)–(2), except conjunction. This is because substituting, say, $X_1 \mapsto X_2 \sqcap X_3$, and repeatedly $X_2 \mapsto X_4 \sqcap X_5$, etc., would yield *infinite branching* in our substitution calculus, which should be carefully dealt with by some restrictions on substitution applications. Instead, we prefer to deal with conjunctions in an incremental fashion, as explained in the next section.

Our tableaux contain concept variables; we denote by $\sigma(\mathbf{T})$ the application of the substitution σ to every concept in every constraint of \mathbf{T} . Since we operate on a *system* of three tableaux, we denote it globally as $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$. For such a system, $\sigma \langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ denotes $\langle \sigma(\mathbf{T}_1), \sigma(\mathbf{T}_2), \sigma(\mathbf{T}_3) \rangle$. When both a T-rule and an S-rule is applicable to $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$, *T-rules have always precedence over S-rules*.

When the application of a rule to $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ yields $\langle \mathbf{T}'_1, \mathbf{T}'_2, \mathbf{T}'_3 \rangle$, we say that $\langle \mathbf{T}'_1, \mathbf{T}'_2, \mathbf{T}'_3 \rangle$ *directly derives from* $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$. Then *derives* is just the transitive closure of “directly derives”. In what follows, we assume that T- and S-rules are applied to three tableaux that always start as follows:

$$\mathbf{T}_1 = \{\mathcal{L}_1(a) = \{C_1, \sim(L_X)\}\} \quad (5)$$

$$\mathbf{T}_2 = \{\mathcal{L}_2(a) = \{C_2, \sim(L_X)\}\} \quad (6)$$

$$\mathbf{T}_3 = \{\mathcal{L}_3(a) = \{L, \sim(L_X)\}\} \quad (7)$$

For such tableaux, the following properties can be isolated.

Lemma 2 *Let $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ start as above. Then, (1) every concept variable occurs always with a negation in front in every constraint of every tableau; (2) every T-complete branch \mathcal{L}_i of every \mathbf{T}_i contains at most one concept variable X such that $\neg X \in \mathcal{L}_i(x)$, for some individual x .*

Proof. Property (1) can be proved by induction on T- and S-rule applications. Base: in the initial tableaux (5)–(7), concept variables appear only in $\sim(L_X)$, and since variables occur positively in L_X , negation normal form puts a “ \neg ” in front of every variable. Induction: suppose the claim holds for $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$. By inspection, T-rules never introduce nor delete negations in concepts, so the claim holds also after a T-rule has been applied. S-rules which introduce new concept

All rules are applicable only if $\mathcal{L} \in \mathbf{T}_1 \cup \mathbf{T}_2$, \mathcal{L} is *open*, and the substitution is *not* σ -*blocked*. Rules above the separating line have precedence over rules below it.

$\sigma\top$ -rule : if $\neg X \in \mathcal{L}(x)$, then apply $\sigma = \{X \mapsto \top\}$ to $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$
σN -rule : if $\{\neg X, A\} \subseteq \mathcal{L}(x)$ for some $A \in N_c$, then apply $\sigma = \{X \mapsto A\}$ to $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$
$\sigma\neg N$ -rule : if $\{\neg X, \neg A\} \in \mathcal{L}(x)$ for some $A \in N_c$, then apply $\sigma = \{X \mapsto \neg A\}$, to $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$
$\sigma \geq$ -rule : if $\neg X \in \mathcal{L}(x)$ and there are exactly n R -neighbors of x , then apply $\sigma = \{X \mapsto (\geq m S)\}$, where m is between 0 and n , and $R \sqsubseteq^* S$
$\sigma \leq$ -rule : if $\{\neg X, (\leq n S)\} \subseteq \mathcal{L}(x)$, then apply $\sigma = \{X \mapsto (\leq n R)\}$ to $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$, for some role R such that $R \sqsubseteq^* S$
<hr/> $\sigma\forall$ -rule : if $\{\neg X, \forall S.C\} \subseteq \mathcal{L}(x)$, then apply $\sigma = \{X \mapsto \forall R.Y\}$ to $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$, where Y denotes a concept variable not appearing in $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$, and $R \sqsubseteq^* S$
$\sigma\exists$ -rule : if $\{\neg X, \exists R.C\} \subseteq \mathcal{L}(x)$, then apply $\sigma = \{X \mapsto \exists S.Y\}$ to $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$, where Y denotes a concept variable not appearing in $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$, and $R \sqsubseteq^* S$

Figure 2: Substitution rules (S-rules) for $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{LN}_{R^+}$

variables ($\sigma\forall$ and $\sigma\exists$) apply a substitution $X \mapsto D$ (where D is either $\forall R.Y$ or $\exists S.Y$) to an existing negated variable $\neg X$ (by induction hypothesis). By Def.3, $\neg X$ is substituted with $\sim D$, so also newly introduced concept variables satisfy the claim.

We now turn to prove (2). At start, concept variables appear only in $\sim(L_X)$, and S-rules never increase the number of concept variables: all S-rules reduce by 1 the number of concept variables, but for $\sigma\forall$ and $\sigma\exists$ that introduce a new variable Y , but remove X . So, we can base our induction on the number n of variables of L_X . If L contains no quantifications (i.e., L is a conjunction of atomic concepts), then $\sim(L_X) = \sim(X_0 \sqcap L) = \neg X_0 \sqcup \sim L$, so the claim holds for one variable $X = X_0$ and $x = a$. Suppose the claim holds for concepts with n variables. If L contains $n + 1$ variables, then it must contain at least one quantification, and L_X can be decomposed as $X_0 \sqcap L_1$, where L_1 is a concept *term* that contains at least one quantification, so another variable, and at most n variables in total. Hence, $\sim(L_X) = \neg X_0 \sqcup \sim(L_1)$, so from $\sim(L_X) \in \mathcal{L}_i(a)$ T-rules can obtain two branches, say, \mathcal{L}'_i and \mathcal{L}''_i , the former with $\neg X_0 \in \mathcal{L}'_i(a)$, and the latter with $\sim(L_1) \in \mathcal{L}''_i(a)$. For \mathcal{L}'_i the claim holds directly, while for \mathcal{L}''_i it holds by inductive hypothesis since it contains n variables. \square

The above lemma justifies the absence of S-rules acting for constraints of the form $X \in \mathcal{L}_i(x)$, since starting from $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ as in (5)–(7), such constraints never appear.

A branch is *S-complete* if no S-rule is applicable. We call a branch *complete* when it is both T-complete and S-complete.

Theorem 3 (Soundness and completeness) *Let C_1, C_2, L as in Thm.2, and let $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ be defined as in (5)–(7). Then Formula (4) is true if and only if there is a way of*

applying S-rules, obtaining a global substitution σ , such that both $\sigma(\mathbf{T}_1)$ and $\sigma(\mathbf{T}_2)$ close, and $\sigma(\mathbf{T}_3)$ is open.

Proof. (Only if.) Validity of Formula (4) requires a ground substitution σ' . Now σ' may not be used to prove directly the claim, since it may contain \sqcap in some substitution $X_i \mapsto D_i$, and \sqcap is not reconstructed by S-rules. So let σ be a substitution obtained from σ' by choosing only one conjunct in the outermost \sqcap of each D_i , and if such a conjunct contains an \sqcap (also inside quantifications), recursively choosing one conjunct, till one obtains a D'_i without \sqcap s; the choice is made by inspecting how T-rules build the branches of the (variable-free) tableau $\sigma'(\mathbf{T}_3)$. Observe that $\sigma'(\sim(L_X)) = \sigma'(\neg X_0) \sqcup \sigma'(\sim(L_1))$ for a suitable concept term L_1 , and suppose that σ' contains the substitution $X_0 \mapsto E_1 \sqcap E_2$. Therefore, \sqcup -rule applied to $\sigma'(\sim(L_X)) \in \mathcal{L}_3(a)$ yields three branches, $\neg E_1 \in \mathcal{L}'_3(a)$, $\neg E_2 \in \mathcal{L}''_3(a)$, and $\sigma'(\sim(L_1)) \in \mathcal{L}'''_3(a)$. Since σ' validates (4), at least one among $\mathcal{L}'_3, \mathcal{L}''_3, \mathcal{L}'''_3$ can be turned by T-rules into a T-complete, open branch. If such a branch is \mathcal{L}'_3 , choose $X_0 \mapsto E_1$ for σ , if it is \mathcal{L}''_3 choose E_2 , while if the open branch stems from \mathcal{L}'''_3 then choose whatever E_1, E_2 , indifferently. Clearly if E_i is chosen and E_i still contains conjunctions, the choice is recursively repeated. Soundness of T-rules ensures that the choice of a \sqcap -free substitution σ can always be made in such a way that, finally, T-rules can obtain from $\sigma(\mathbf{T}_3)$ a T-complete, open branch. Observe also that $\sigma'(\mathbf{T}_1), \sigma'(\mathbf{T}_2)$ must close by completeness of T-rules, that is, every branch stemming from them must close. Now branches from $\sigma(\mathbf{T}_1)$ and $\sigma(\mathbf{T}_2)$ are a subset of the branches from $\sigma'(\mathbf{T}_1), \sigma'(\mathbf{T}_2)$, hence all of them must close too. It remains to show that σ can be reconstructed by repeated application of S-rules, and which can be proved by induction on the quantifications of each D_i in $X_i \mapsto D_i \in \sigma$.

(If.) If S-rules (intertwined with T-rules) can construct a ground substitution σ such that both $\sigma(\mathbf{T}_1)$ and $\sigma(\mathbf{T}_2)$ close, and $\sigma(\mathbf{T}_3)$ is open, then by soundness of T-rules, σ is a witness validating Formula (4). \square

The above theorem does not exclude that, when Formula (4) is false, the calculus runs forever. In fact, the reader could verify that in the example *trans(P)*, $C_1 = \exists P \sqcap \forall P. \exists P.(A \sqcap C)$, $C_2 = \exists P \sqcap \forall P. \exists P.(B \sqcap C)$, and $L = \exists P \sqcap \forall P. \exists P.C$ T- and S-rules together run indefinitely. Intuitively, the calculus can go astray when already $L \in LCS(C_1, C_2)$, and transitive roles keep producing new individuals and concepts that, in turn, trigger the application of an S-rule, which may add new concepts to old individuals, and such concepts can propagate to new individuals, destroying pairwise blocking. Therefore, although S-rules require some other constraints to be already present in the branch, they also need a *blocking condition*, to prevent their infinite application.

A substitution $X \mapsto \exists R.Y$ is *S-blocked* for $\neg X \in \mathcal{L}_i(x)$ in $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ if $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ derives from some $\langle \mathbf{T}'_1, \mathbf{T}'_2, \mathbf{T}'_3 \rangle$, in which there is some individual x' such that: (i) $\neg X' \in \mathcal{L}'_i(x')$, (ii) $\mathcal{L}_i(x) = \mathcal{L}'_i(x')$, (iii) for every R -successor y of x in \mathcal{L}_i , there exists an R -successor y' of x' in \mathcal{L}'_i such that $\mathcal{L}_i(y) = \mathcal{L}'_i(y')$, (iv) for every S , the number of different S -neighbors of x in \mathcal{L}_i is the same as the number of different S -neighbors of x' in \mathcal{L}'_i , and (v) the $\sigma\exists$ -rule has been

applied to $\langle \mathbf{T}'_1, \mathbf{T}'_2, \mathbf{T}'_3 \rangle$, with the substitution $X' \mapsto \exists R.Y'$. The S-blocking of a substitution $X \mapsto \forall R.Y$ is defined analogously. Observe that we compare $\mathcal{L}_i(x)$ (the concepts attached to x in $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$) with $\mathcal{L}'_i(x')$ (that is, the concepts attached to x' in the old state $\langle \mathbf{T}'_1, \mathbf{T}'_2, \mathbf{T}'_3 \rangle$). Also, note that Lemma 2 allows us to define blocking only for constraints of the form $\neg X \in \mathcal{L}_i(x)$.

Rule $\sigma\exists$ is S-blocked for $\neg X \in \mathcal{L}_i(x)$ in $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ if either it is simply not applicable, or every possible substitution for X allowed by $\sigma\exists$ is S-blocked, and analogously for Rule $\sigma\forall$. Note that if $\{\neg X, \exists R.C\} \subseteq \mathcal{L}(x)$, then for each role S such that $R \sqsubseteq^* S$, Rule $\sigma\exists$ allows the substitution $X \mapsto \exists S.Y$, so S-blocking prevents all these substitutions. Finally, we say that a *branch is S-blocked* if it is not closed and contains a constraint $\neg X \in \mathcal{L}_i(x)$ for which both Rule $\sigma\exists$ and Rule $\sigma\forall$ are S-blocked, and $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ is S-blocked if either \mathbf{T}_1 or \mathbf{T}_2 contain an S-blocked branch.

Now we modify the completeness of a branch by saying that a branch is S-complete if no S-rule is applicable, taking also S-blocking into account. Clearly, when we stop the calculus because of S-blocking, we have to prove that no substitution was ever to be found.

Theorem 4 *If $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ is S-blocked, no $\langle \mathbf{T}'_1, \mathbf{T}'_2, \mathbf{T}'_3 \rangle$ such that \mathbf{T}'_1 and \mathbf{T}'_2 are closed, and \mathbf{T}'_3 is open, can be derived from it.*

Proof. (Sketch) We intuitively view our calculus as a game between Player T, whose moves are T-rules, and player S, whose moves are S-rules. Faithfully to our precedences, T moves whenever she can, S moves only if T cannot move, and S should use a move above the line in Fig. 2 whenever he can. S wins if he can reach a state $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ in which both \mathbf{T}_1 and \mathbf{T}_2 are closed, and \mathbf{T}_3 is open, while T wins in every other case (including infinite runs). Intuitively, S tries to build some finite proof of (4), while T responds by constructing a (possibly infinite) model that would serve as a counterexample for that proof. In this setting, the claim is proved if T has a winning strategy whenever $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ is S-blocked.

In fact, suppose that in such a case S tries anyway a substitution $X \mapsto D$ for $\neg X \in \mathcal{L}_i(x)$. Then, T can respond in the same way she did in $\langle \mathbf{T}'_1, \mathbf{T}'_2, \mathbf{T}'_3 \rangle$ when S played $X' \mapsto D'$ for $\neg X' \in \mathcal{L}'_i(x')$. Conditions (ii)–(iv) ensure that after S plays $X \mapsto D$ in $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$, T can play on x the same rules she played on x' after S played $X' \mapsto D'$ in $\langle \mathbf{T}'_1, \mathbf{T}'_2, \mathbf{T}'_3 \rangle$. S will not succeed in closing \mathcal{L}_i , otherwise by precedence of Rules $\sigma\top$, $\sigma\mathbf{N}$, $\sigma\rightarrow\mathbf{N}$, $\sigma\geq$, $\sigma\leq$, over Rules $\sigma\forall$ and $\sigma\exists$, S would have closed \mathcal{L}'_i in $\langle \mathbf{T}'_1, \mathbf{T}'_2, \mathbf{T}'_3 \rangle$ before deriving $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$. \square

Theorem 5 (Termination) *Let T-rules and S-rules be applied according to blocking conditions, giving always precedence to T-rules. Then there is no infinite sequence of applications of T- and S-rules starting from $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ as in (5)–(7).*

Proof. (Sketch) Termination of T-rules alone was proved by Tobies [2001] (Lemma 6.35). Termination of T- and S-rules together stems from S-blocking, which eventually occurs since S-rules add concepts that are in the syntactic closure of C_1, C_2, L and \mathcal{H} . In fact, observe that, e.g., $\sigma\exists$ substitutes X with $\exists S.Y$ only if $\exists R.C \in \mathcal{L}_i(x)$, and $\exists R.C$ does not

come from another substitution in the same branch, because of Lemma 2. \square

In conclusion, checking whether $L \in LCS(C_1, C_2)$ is a decidable problem for $C_1, C_2, L \in \mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$.

6 Computing the LCS

The previous section set up a calculus for the LCS *decision* problem. We now set an iterative algorithm that computes the LCS by repeatedly solving Formula (4) for increasingly better L s.

Algorithm 2 Computing an LCS of C_1, C_2

```

input concepts  $C_1, C_2$ 
var concept  $L := \top$ , concept  $L_1$ 
repeat (*)
   $\mathbf{T}_1 := \{C_1 \in \mathcal{L}_1(a), \sim(L_X) \in \mathcal{L}_1(a)\}$ ;
   $\mathbf{T}_2 := \{C_2 \in \mathcal{L}_2(a), \sim(L_X) \in \mathcal{L}_2(a)\}$ ;
   $\mathbf{T}_3 := \{L \in \mathcal{L}_3(a), \sim(L_X) \in \mathcal{L}_3(a)\}$ ;
  apply T-rules and S-rules to  $\langle \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3 \rangle$ 
  if a substitution  $\sigma$  s.t. ( $\mathbf{T}_1, \mathbf{T}_2$  close and  $\mathbf{T}_3$  is open) is found
    then  $L_1 := \sigma(L_X)$ ;  $L := L_1$ 
    else  $L_1 := nil$ ;
until ( $L_1 = nil$ )
return  $L$ ;
```

Termination of the above algorithm implies the existence of a finite LCS for every pair of concepts in $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$. This problem is out of the scope of this paper, hence we give a weaker termination proof.

Theorem 6 *Let $C_1, C_2 \in \mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{I}\mathcal{N}_{R^+}$. If $LCS(C_1, C_2)$ contains a finite concept expression, then Algorithm 2 terminates with $L \in LCS(C_1, C_2)$.*

Proof. If there exists a finite concept $\hat{L} \in LCS(C_1, C_2)$, it must have a finite number of non-redundant conjuncts. In each iteration (*), in order for \mathbf{T}_3 to be open, $L_1 = \sigma(L_X)$ must have at least one non-redundant conjunct more than L . Hence after $|\hat{L}|$ iterations, the **until** condition is reached. \square

We remark that for DLs for which a finite LCS always exists, the above theorem implies that Algorithm 2 always terminates. For instance, in $\mathcal{AL}\mathcal{E}\mathcal{N}$ there always exist a finite $LCS(C_1, C_2)$, whose size is exponential in the sizes of C_1, C_2 [Küsters and Molitor, 2005]. Hence for $C_1, C_2 \in \mathcal{AL}\mathcal{E}\mathcal{N}$, Algorithm 2 iterates (*) a number of times exponential in $|C_1| + |C_2|$.

Also, we remark that if iterations (*) are stopped before the **until** condition is true, an invariant of (*) is that the current value of L is a common subsumer of C_1, C_2 (although not the least one). In this sense, Algorithm 2 is also an *anytime approximation* algorithm for LCS.

7 Conclusion and Perspective

Although Least Common Subsumer is one of the most interesting and usefully exploited non-standard inference service for Description Logics, its computation for expressive DLs is still an open challenge.

In this paper we formulated the problem of evaluating LCS in terms of second-order formulas where variables represent general DL concepts. Based on this formulation we also proposed a novel general tableau-based calculus to compute a solution to a LCS problem and presented the whole calculus for an expressive DL, namely $\mathcal{AL}\mathcal{E}\mathcal{H}\mathcal{LN}_{R^+}$.

Having a calculus based on well-founded analytic tableaux surely presents many advantages both from a practical point of view and from a theoretical one. First, our approach may ease implementing the computation of LCS in state-of-the-art tableaux-based reasoners (Pellet, FaCT++, RacerPro), also exploiting well known optimization techniques for tableaux in DLs [Baader *et al.*, 2003, Ch.9]. Secondly, the analysis of soundness and completeness for a tableau-based algorithm is less tricky than the one based on structural algorithms as the ones proposed so far for LCS. Finally, but even more important, for DLs in which a finite LCS may not always exist [Baader *et al.*, 2007], a terminating algorithm for computing LCS cannot exist, while a sound and complete calculus can be devised along the lines we showed—analogously to sound and complete calculi for full First-Order Logic.

In perspective, our formulation and computation of LCS paves the way to further results for computing other useful non-standard reasoning services in DLs.

Acknowledgments

We thank Franz Baader for useful discussions on LCS and pointers to his relevant literature, and all reviewers for thorough revisions and suggestions. We acknowledge partial support of Apulia region Strategic Projects PS_092 and PS_121.

References

- [Baader and Küsters, 1998] F. Baader and R. Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic \mathcal{ALN} -concept descriptions. In *Proc. of KI'98*, p. 129–140, 1998. Springer.
- [Baader and Narendran, 2001] F. Baader and P. Narendran. Unification of concept terms in description logics. *J. of Symbolic Computation*, 31:277–305, 2001.
- [Baader and Turhan, 2002] F. Baader and A. Y. Turhan. On the problem of computing small representations of least common subsumers. In *Proc. of KI 2002*, 2002. Springer.
- [Baader *et al.*, 1999] F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proc. of IJCAI'99*, p. 96–101. Morgan Kaufmann, 1999.
- [Baader *et al.*, 2003] F. Baader, D. Calvanese, D. Mc Guinness, D. Nardi, and P. Patel-Schneider, eds. *The Description Logic Handbook*. Cambridge University Press, 2003.
- [Baader *et al.*, 2007] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. of Appl. Log.*, 5(3):392–420, 2007.
- [Benatallah *et al.*, 2005] B. Benatallah, M. S. Hacid, A. Leger, C. Rey, and F. Toumani. On automating web services discovery. *VLDB Journal*, 2005.
- [Brandt *et al.*, 2003] S. Brandt, A.-Y. Turhan, and R. Küsters. Foundations of non-standard inferences for description logics with transitive roles and role hierarchies. LTCS-Report 03-02, 2003.
- [Cohen and Hirsh, 1994] W. W. Cohen and H. Hirsh. Learning the CLASSIC description logics: Theoretical and experimental results. In *Proc. of KR'94*, p. 121–133, 1994.
- [Cohen *et al.*, 1992] W. Cohen, A. Borgida, and H. Hirsh. Computing least common subsumers in description logics. In *Proc. of AAAI'92*, p. 754–761, 1992. AAAI Press.
- [Colucci *et al.*, 2008] S. Colucci, E. Di Sciascio, F. M. Donini, and E. Tinelli. Finding informative commonalities in concept collections. In *Proc. of CIKM 2008*, p. 807–816. ACM Press, 2008.
- [Hacid *et al.*, 2000] M. S. Hacid, F. Soualmia, and F. Toumani. Schema Extraction for Semistructured Data. In *Proc. of DL 2000*, 2000.
- [Hacid *et al.*, 2002] M. S. Hacid, A. Leger, and C. Rey. Computing concept covers: A preliminary report. In *Proc. of DL 2002*, 2002.
- [Henkin, 1950] L. Henkin. Completeness in the theory of types. *J. Symb. Log.*, 15(2):81–91, 1950.
- [Janowicz *et al.*, 2008] K. Janowicz, M. Wilkes, and M. Lutz. Similarity-based information retrieval and its role within spatial data infrastructures. In *Proc. of GIScience '08*, p. 151–167. Springer, 2008.
- [Karam *et al.*, 2007] N. Karam, S. Linckels, and C. Meinel. Semantic composition of lecture subparts for a personalized e-learning. In *Proc. of ESWC 2007*, p. 716–728. Springer, 2007.
- [Kopena and Regli, 2003] J. B. Kopena and W. C. Regli. Design repositories on the semantic web with description-logic enabled services. In *Proc. of VLDB 2003*, 2003.
- [Küsters and Molitor, 2005] R. Küsters and R. Molitor. Structural Subsumption and Least Common Subsumers in a Description Logic with Existential and Number Restrictions. *Studia Logica*, 81:227–259, 2005.
- [Lutz *et al.*, 2006] C. Lutz, F. Baader, E. Franconi, D. Lembo, R. Möller, R. Rosati, U. Sattler, B. Suntisrivaraporn, and S. Tessaris. Reasoning support for ontology design. In *Proc. of OWLED 2006*, 2006.
- [Möller *et al.*, 1998] R. Möller, V. Haarslev, and B. Neumann. Semantics-based information retrieval. In *Proc. of IT&KNOWS-98*, 1998.
- [Ovchinnikova *et al.*, 2007] E. Ovchinnikova, T. Wandmacher, and K. U. Kuehnberger. Solving terminological inconsistency problems in ontology design. *Interoperability in Business Information Systems*, 2:65–80, 2007.
- [Rabin, 1969] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. of the Am. Math. Soc.*, 141:1–35, 1969.
- [Tobies, 2001] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.