Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

The Incredible ELK

From Polynomial Procedures to Efficient Reasoning with $\mathcal{EL}$ Ontologies

Yevgeny Kazakov · **Markus Krötzsch** · **František Simančík**

Received: date / Accepted: date

Abstract $\mathcal{EL}$ is a simple tractable Description Logic that features conjunctions and existential restrictions. Due to its favorable computational properties and relevance to existing ontologies, $\mathcal{EL}$ has become the language of choice for terminological reasoning in biomedical applications, and has formed the basis of the OWL EL profile of the Web ontology language OWL. This paper describes ELK—a high performance reasoner for OWL EL ontologies—and details various aspects from theory to implementation that make ELK one of the most competitive reasoning systems for $\mathcal{EL}$ ontologies available today.

Keywords Description Logics · Implementation and Optimization Techniques · Saturation Procedures · Concurrency

1 Introduction

One of the central research goals in Description Logics (DLs) [10] is finding knowledge representation languages with the right balance between expressivity—what can be said using the language—and complexity—how difficult it is to check if something specific holds. For the lack of a better formal criterion, 'complexity' of DLs was commonly measured in terms of the worst case algorithmic behavior. This research area has resulted in discovery and classification of a broad spectrum of DLs, from simple tractable languages, such as $\mathcal{EL}$ [8] and DL-Lite [25], to very expressive languages of high computational complexity, such as $\mathcal{SROIQ}$ [45,50].[1]

Yevgeny Kazakov
Institute of Artificial Intelligence, University of Ulm, Germany
E-mail: yevgeny.kazakov@uni-ulm.de

Markus Krötzsch
Department of Computer Science, University of Oxford, UK
E-mail: markus.kroetzsch@cs.ox.ac.uk

František Simančík
Department of Computer Science, University of Oxford, UK
E-mail: frantisek.simancik@cs.ox.ac.uk

[1] See the *Description Logic Complexity Navigator* for a comprehensive overview of decidability and complexity results in DLs: http://www.cs.man.ac.uk/~ezolin/dl/

Amongst many languages, the $\mathcal{EL}$ family of DLs has recently received significant interest. $\mathcal{EL}$ is a simple DL which features the top concept ($\top$), conjunctions ($C \sqcap D$), and existential restrictions ($\exists R.C$) as the only concept constructors. These, however, are some of the most common constructors used in existing ontologies. Some of today's largest ontologies, such as SNOMED CT [92], a medical ontology describing about 300,000 concepts, OpenGALEN [84], a medical ontology describing about 23,000 concepts, and many Open Biomedical Ontologies (OBO) [97] can almost completely be expressed in $\mathcal{EL}$.

Biology and medicine are particularly prominent application areas of DLs due to a large number of technical terms involved. To deal effectively with large vocabularies, concepts are typically organized in a hierarchical structure called a *taxonomy* which reflects the *subsumption (is-a) relation* between them. In a medical ontology, e.g., the concept 'Flu' would be subsumed by the concept 'Disease' and would be placed under this concept in the taxonomy. DL-based ontology languages help to reduce redundancies in modeling of taxonomies: rather than stating all relations between concepts explicitly, ontology engineers provide definitions of concepts and their general properties, from which the subsumption relations can be computed using an appropriate reasoning procedure [87]. Reasoning also plays an important role during the design of ontologies (e.g., for detecting inconsistencies and other modeling errors [85]) and in the deployment of ontologies (e.g., for query answering [81]).

The main terminological reasoning problem in DLs is *ontology classification*, whose goal is to compute the taxonomy. Other reasoning problems include checking consistency of the ontology, checking satisfiability of a (complex) concept, and checking whether one (complex) concept is subsumed by another. It turns out that $\mathcal{EL}$ is robustly tractable for these reasoning problems: not only are these problems polynomially solvable for $\mathcal{EL}$,[2] but this holds even if the language is augmented with cyclic definitions, general concept inclusion axioms (GCIs), role hierarchies, complex role inclusion axioms, nominals (and thus ABox assertions), bottom concept, and some forms of datatype restrictions and role range restrictions [6,8,9,23]. These extensions of $\mathcal{EL}$ are often called *the $\mathcal{EL}$ family of DLs*. We will often, however, omit the 'family' suffix when we refer to such extensions. Thus, unless specified otherwise, '$\mathcal{EL}$ ontologies' and '$\mathcal{EL}$ reasoners' will be understood in a broad sense as ontologies expressed in the $\mathcal{EL}$ family of DLs and reasoners for such ontologies.

Polynomial complexity results for $\mathcal{EL}$ can be regarded as a strong indication that the problems can be solved efficiently in practice, but it is not a guarantee. For example, if we perform (quadratically many) subsumption tests between every pair of the 300,000 concepts in SNOMED CT to compute the classification, and every test takes just a constant time, say 1 millisecond, then it will take an estimated 25,000 hours (over 2.8 years) to compute all subsumptions. Clearly, this procedure, although polynomial, can hardly be regarded as practical. There is a significant difference between procedures that perform quadratically many operations for all (or typical) inputs, and procedures that perform quadratically many operations only in some pathological cases, which are unlikely to occur in practice. This is one of the main reasons why highly optimized (tableau-based) procedures can perform very well in practice despite often very high complexity of the languages involved [45,46,76].

The $\mathcal{EL}$ classification procedures, however, have several other strong indicators pointing to a good practical performance. Unlike conventional *tableau-based procedures* [46], which test unknown subsumptions by trying to construct counter-models, the $\mathcal{EL}$ procedures derive new subsumptions explicitly using inference rules. This has two important consequences. First, the reasoner never inspects subsumptions that are not entailed by the ontology. The

[2] The ontology and concept consistency problems are, in fact, trivial in $\mathcal{EL}$ since this language is too weak to express inconsistencies.

number of entailed subsumptions is typically much smaller than the number of all pairs of concepts. For example, SNOMED CT entails only about 5 million subsumption relations, which is less than 0.01% of the total number of possible subsumptions. Second, the $\mathcal{EL}$ classification procedure computes all subsumptions at once in 'one pass', which requires fewer operations than testing the same number of subsumptions separately.

Although modern tableau-based reasoners, such as HermiT, FaCT++, Pellet, and RacerPro, incorporate many optimizations that can reduce the number of subsumption tests and reuse the results of computations between the tests [34,74,104], they still cannot achieve the performance of specialized $\mathcal{EL}$ reasoners on $\mathcal{EL}$ ontologies. For example, the $\mathcal{EL}$ version of OpenGALEN cannot be classified by any tableau reasoners available today, but can be classified by all existing $\mathcal{EL}$ reasoners. The main difficulty for tableau reasoners is that OpenGALEN contains many cyclic axioms, which result in very large models.

Of course, $\mathcal{EL}$ reasoners have the advantage of dealing with a much simpler language, so it may seem unfair to compare specialized $\mathcal{EL}$ procedures with general-purpose ones. It has been recently shown, however, that $\mathcal{EL}$-style classification procedures are not limited to just $\mathcal{EL}$, or even to tractable DLs, but can be extended to more expressive DLs, such as Horn-$\mathcal{SHIQ}$ [51] and (non-Horn) $\mathcal{ALCH}$, [95] while preserving the mentioned properties. These so-called *consequence-based procedures* have other distinguished properties, such as (i) optimal worst-case complexity, (ii) 'pay-as-you-go' behavior: the more $\mathcal{EL}$ constructors ontology uses, the more it behaves like the $\mathcal{EL}$ procedure, and (iii) determinism: the procedure does not make choices or backtracking even for DLs with disjunctions, such as $\mathcal{ALCH}$.

This paper describes ELK—an open source, Java-based reasoner for OWL EL ontologies.[3] OWL EL is a profile of the W3C standardized logic-based ontology language OWL [75,82] based on the $\mathcal{EL}$ family of DLs. The main goals of ELK are extensive coverage of OWL EL features, high performance of reasoning, and easy extensibility and use. Since its first release in 2011, ELK has already been used in a variety of biomedical applications, e.g., to model the neuroanatomy of flies [81], to integrate databases of diseases, genes, and drugs [39,40,37], and to validate and query genetic ontologies [48,101]. As of this paper, the latest release 0.3.2 of ELK supports a fragment of OWL EL that corresponds to the DL $\mathcal{EL}_\bot^+$, which additionally features the bottom concept ($\bot$) and (complex) role inclusion axioms.

Although the procedure in ELK shares many similarities with existing $\mathcal{EL}$ procedures [8], it offers a range of significant improvements. Firstly, the procedure applies inference rules in a goal-directed way and avoids redundant inferences without compromising completeness. Secondly, the procedure is able to apply inferences in parallel, which can take advantage of existing multiprocessor systems. In combination with some further implementation techniques, such as indexing and efficient join computation, these improvements result in a significant performance increase compared with other $\mathcal{EL}$ reasoners. For example, SNOMED CT can be classified in about 10 minutes by the $\mathcal{EL}$ reasoners CEL [14] and jcel [72], and in about 25 seconds by the $\mathcal{EL}$ reasoner Snorocket [66]. The same ontology can now be classified by ELK in as little as 5 seconds on the same (quad-core) computer.

In detail, the main contributions of this paper can be summarized as follows:

1. We present a new rule-based procedure for reasoning in $\mathcal{EL}_\bot^+$ that incorporates many enhancements and optimizations compared to the previously proposed reasoning procedures. In particular, the procedure does not require the input ontology to be normalized, and has a novel redundancy condition that can be used to avoid unnecessary inferences.
2. We present a new method for concurrent application of inference rules, which can be used with arbitrary rule systems, and in particular, with our $\mathcal{EL}_\bot^+$ procedure. The method

[3] http://elk.semanticweb.org/

assigns expressions participating in the rules to one or more ‘contexts’ such that inferences are possible only between elements of the same context. This way, inferences in different contexts can be performed in parallel. Another advantage of our method is that it does not require the datastructures for storing conclusions of the rules to be thread-safe.
3. We describe some new implementation techniques, which contribute to the improved reasoning performance of ELK. The techniques include indexing of axioms for efficient rule application, optimized join evaluation for premises of the rules, caching of partial joins, practical implementation of redundancy, and optimized transitive reduction.
4. We provide an extensive experimental evaluation measuring the effect of concurrency and other optimizations in ELK on a collection of some of the largest $\mathcal{EL}_\bot^+$ ontologies that we were able to obtain from public and commercial sources. We compare the improvements both in system-dependent values, such as running times, as well as system-independent values, such as the number of rule applications.

Some results presented in this paper were previously published in conference and workshop proceedings. In particular, the concurrent procedure for $\mathcal{EL}_\bot^+$ ontologies [53] and implementation details of the ELK reasoner [55]. This paper should be self-contained and does not require any prior knowledge in Description Logics, OWL, or programming languages. We tried to present a coherent view of the main aspects of the $\mathcal{EL}$ reasoning, from theory to implementation, and provide many examples to illustrate those aspects. We hope, therefore, that the paper can be valuable to those wishing to understand the underlying ideas of the ELK system, and to those wishing to implement similar systems.

The paper is organized as follows. After a brief introduction to the description logic $\mathcal{EL}_\bot^+$, in Section 3 we present a consequence-based calculus for terminological reasoning in $\mathcal{EL}_\bot^+$, prove its soundness and completeness, and gradually optimize it with appropriate notions of redundancy for inferences and goal-directed strategies of rule application. In Section 4 we describe a multi-phase procedure for implementing this calculus using indexing and the abstract saturation procedure, in Section 5 we present several optimizations for this procedure, and in Section 6 we extend our approach to concurrency. Section 7 gives an overview of the ELK system, and Section 8 provides empirical evaluations of optimizations and concurrency. We discuss related works in Section 9 and conclude in Section 10. In Appendix A we describe how our procedure can be used for reasoning with ABox assertions and some restricted types of nominals.

2 Description Logic Preliminaries

Here we introduce the description logic $\mathcal{EL}_\bot^+$, which corresponds to the fragment of OWL EL supported by ELK 0.3.2. Readers who are not familiar with description logics (DLs) may wish to consult a more gentle first introduction [65].

$\mathcal{EL}_\bot^+$ is defined w.r.t. a vocabulary consisting of countably infinite sets of *(atomic) roles*, *atomic concepts*, and *(named) individuals*. Complex *concepts* and *axioms* are defined recursively in Table 1. We use the letters R,S for roles, C,D,E for concepts, A,B for atomic concepts, and a,b,c for individuals. A *concept equivalence* $C \equiv D$ abbreviates the two concept inclusions $C \sqsubseteq D$ and $D \sqsubseteq C$. Axioms that involve individuals are called *assertional axioms*, or *ABox axioms*. Other axioms are called *terminological axioms*, or *TBox axioms*. An *ontology* is a finite set of axioms.

$\mathcal{EL}_\bot^+$ has a Tarski-style semantics. An *interpretation* $\mathcal{I}$ consists of a nonempty set $\Delta^\mathcal{I}$ called the *domain* of $\mathcal{I}$ and an interpretation function $\cdot^\mathcal{I}$ that assigns to each role R a binary

Table 1 Syntax and semantics of $\mathcal{EL}^+_\bot$

	Syntax	Semantics
Roles:		
atomic role	R	$R^\mathcal{I}$
Concepts:		
atomic concept	A	$A^\mathcal{I}$
top	$\top$	$\Delta^\mathcal{I}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$C^\mathcal{I} \cap D^\mathcal{I}$
existential restriction	$\exists R.C$	$\{x \mid \exists y : \langle x,y\rangle \in R^\mathcal{I} \wedge y \in C^\mathcal{I}\}$
Individuals:		
named individual	a	$a^\mathcal{I}$
Axioms:		
concept inclusion	$C \sqsubseteq D$	$C^\mathcal{I} \subseteq D^\mathcal{I}$
role inclusion	$R \sqsubseteq S$	$R^\mathcal{I} \subseteq S^\mathcal{I}$
role composition	$R_1 \circ R_2 \sqsubseteq S$	$\langle x,y\rangle \in R_1^\mathcal{I} \wedge \langle y,z\rangle \in R_2^\mathcal{I} \to \langle x,z\rangle \in S^\mathcal{I}$
concept assertion	$C(a)$	$a^\mathcal{I} \in C^\mathcal{I}$
role assertion	$R(a,b)$	$\langle a^\mathcal{I}, b^\mathcal{I}\rangle \in R^\mathcal{I}$

relation $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$, to each atomic concept A a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$, and to each individual a an element $a^\mathcal{I} \in \Delta^\mathcal{I}$. This assignment is extended to complex concepts as shown in Table 1.

An interpretation $\mathcal{I}$ *satisfies* an axiom α (written $\mathcal{I} \models \alpha$) if the corresponding condition in Table 1 holds. $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ (written $\mathcal{I} \models \mathcal{O}$) if $\mathcal{I}$ satisfies all axioms in $\mathcal{O}$. An ontology is *consistent* if it has at least one model, otherwise it is *inconsistent*. We say that an axiom α is a *consequence* of an ontology $\mathcal{O}$, or also that $\mathcal{O}$ *entails* α (written $\mathcal{O} \models \alpha$), if every model of $\mathcal{O}$ satisfies α. Note that an inconsistent ontology entails every axiom. A concept C is *unsatisfiable* w.r.t. $\mathcal{O}$ if $\mathcal{O} \models C \sqsubseteq \bot$, otherwise C is *satisfiable* w.r.t. $\mathcal{O}$. A concept C is *subsumed* by D w.r.t. $\mathcal{O}$ if $\mathcal{O} \models C \sqsubseteq D$. Concepts C and D are *equivalent* w.r.t. $\mathcal{O}$ if $\mathcal{O} \models C \equiv D$. An individual a is an *instance* of a concept C w.r.t. $\mathcal{O}$ if $\mathcal{O} \models C(a)$.

A general *reasoning problem* in $\mathcal{EL}^+_\bot$ is checking entailment of axioms from ontologies: given an ontology $\mathcal{O}$ and an axiom α, check if $\mathcal{O} \models \alpha$. If both $\mathcal{O}$ and α consist only of terminological axioms, we speak about *terminological reasoning*. In practice one often does not check entailment of a single axiom, but performs a *reasoning task* that consists of checking multiple entailments at once. The goal of the *ontology classification task* is to compute the *taxonomy* representing all entailed subsumptions and equivalences between $\top$, $\bot$, and atomic concepts occurring in $\mathcal{O}$. The goal of the *ontology realization task* is to compute all entailed instances of atomic concepts occurring in $\mathcal{O}$.

To simplify the presentation, we only consider terminological reasoning in the main part of this paper. Thus, without further qualification, we assume that ontologies contain only terminological (TBox) axioms. However, we will show in Appendix A that our algorithms can very easily be adjusted to also support assertional (ABox) axioms and, more generally, certain 'safe' occurrences of nominals (singleton concepts). As we will show, this is essentially because safe nominals are indistinguishable from atomic concepts.

3 Inference Rules for Reasoning in $\mathcal{EL}^+_\bot$

In this section we develop an optimized calculus for reasoning in $\mathcal{EL}^+_\bot$ that is the basis of the procedure implemented in ELK. In Section 3.1 we start with the most simple and least restricted version of the calculus that derives all subsumptions $C \sqsubseteq D$ between arbitrary

$$\mathsf{R}_0\ \frac{}{C \sqsubseteq C} \qquad \mathsf{R}_\top\ \frac{}{C \sqsubseteq \top} \qquad \mathsf{R}_\bot\ \frac{E \sqsubseteq \exists R.C \quad C \sqsubseteq \bot}{E \sqsubseteq \bot}$$

$$\mathsf{R}_\sqcap^-\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2} \qquad \mathsf{R}_\sqcap^+\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} \qquad \mathsf{R}_\exists\ \frac{E \sqsubseteq \exists R.C \quad C \sqsubseteq D}{E \sqsubseteq \exists R.D}$$

$$\mathsf{R}_\sqsubseteq\ \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O} \qquad \mathsf{R}_\mathsf{H}\ \frac{E \sqsubseteq \exists R.C}{E \sqsubseteq \exists S.C} : R \sqsubseteq S \in \mathcal{O} \qquad \mathsf{R}_\circ\ \frac{E \sqsubseteq \exists R_1.C \quad C \sqsubseteq \exists R_2.D}{E \sqsubseteq \exists S.D} : R_1 \circ R_2 \sqsubseteq S \in \mathcal{O}$$

Fig. 1 Basic inference rules for reasoning in $\mathcal{EL}_\bot^+$

(complex) $\mathcal{EL}_\bot^+$ concepts C and D entailed from the given $\mathcal{EL}_\bot^+$ ontology. In Section 3.2 we formulate a more restricted calculus in which existential conclusions of some inferences are distinguished, and demonstrate how this helps to avoid some rule applications using a new redundancy condition. In Section 3.3 we prove that the restricted calculus does not loose any of the entailed subsumptions—some subsumptions will just have fewer different proofs. In Section 3.4 we show how to further restrict the rules if one is interested not in all derivable subsumptions, but only in some selected subset of (goal) subsumptions. In Section 3.5 we then show how this result can be used to obtain a polynomial-time classification procedure for $\mathcal{EL}_\bot^+$. While the specific calculus we develop here is new, many similar calculi have been studied in the literature. We give an overview of such related works in Section 3.6.

3.1 The Basic Inference Rules

In this section we present a simple procedure for reasoning in $\mathcal{EL}_\bot^+$, which we are going to refine gradually. The calculus works by applying the rules in Fig. 1, to derive subsumptions between $\mathcal{EL}_\bot^+$ concepts. We distinguish between the premises of a rule (above the horizontal line), its conclusions (below the horizontal line), and its side conditions (after the colon). Rules R_0 and $\mathsf{R}_\top$ have no premises and (so far) can be used with arbitrary concepts C. Note that the axioms in $\mathcal{O}$ are only used as side conditions of rules $\mathsf{R}_\sqsubseteq$, R_H, and $\mathsf{R}_\circ$.

Intuitively, rules R_0 and $\mathsf{R}_\top$ derive trivial subsumptions. Rule $\mathsf{R}_\bot$ propagates inconsistency of fillers in existential restrictions. Rules $\mathsf{R}_\sqcap^-$ and $\mathsf{R}_\sqcap^+$ decompose and compose conjunctions on the right-hand side. Rule $\mathsf{R}_\exists$ replaces the filler of the existential restriction on the right-hand side using a derived subsumption. Rule $\mathsf{R}_\sqsubseteq$ unfolds (told) subsumptions in the ontology. Rules R_H and $\mathsf{R}_\circ$ use role inclusion and composition axioms to produce new existential restrictions. Note that no rule creates a complex concept on the left-hand side of subsumptions. This is different from typical sequence-based rules. Also note that rule $\mathsf{R}_\exists$ uses a (derived) subsumption $C \sqsubseteq D$, and not a told subsumption from the ontology, like $D \sqsubseteq E$ in $\mathsf{R}_\sqsubseteq$. From the following example one can see that if the second premise of $\mathsf{R}_\exists$ is likewise restricted to only told subsumptions, some subsumptions would not be derivable.

Example 1 Consider the ontology $\mathcal{O}$ consisting of the following axioms:

(ax1): $A \sqsubseteq \exists R.(C \sqcap D)$ (ax2): $B \equiv A \sqcap \exists S.D$ (ax3): $\exists S.D \sqsubseteq C$ (ax4): $R \sqsubseteq S$.

The rules in Fig. 1 can be used to derive the subsumption $A \sqsubseteq B$ using axioms (ax1)–(ax4).

$$A \sqsubseteq A \qquad \text{by } \mathsf{R}_0 \qquad (1)$$

$$A \sqsubseteq \exists R.(C \sqcap D) \qquad \text{by } \mathsf{R}_\sqsubseteq \text{ to (1) using (ax1)} \qquad (2)$$

$$C \sqcap D \sqsubseteq C \sqcap D \qquad \text{by } \mathsf{R}_0 \qquad (3)$$

$$\mathsf{R}_0\ \frac{}{C \sqsubseteq C} \qquad \mathsf{R}_\top\ \frac{}{C \sqsubseteq \top} \qquad \mathsf{R}_\bot\ \frac{E \xrightarrow{R} C \quad C \sqsubseteq \bot}{E \sqsubseteq \bot}$$

$$\mathsf{R}_\sqcap^-\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2} \qquad \mathsf{R}_\sqcap^+\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2}$$

$$\mathsf{R}_\exists^-\ \frac{E \sqsubseteq \exists R.C}{E \xrightarrow{R} C} \qquad \mathsf{R}_\exists^+\ \frac{E \xrightarrow{R} C \quad C \sqsubseteq D}{E \sqsubseteq \exists R.D}$$

$$\mathsf{R}_\sqsubseteq\ \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O} \qquad \mathsf{R}_\mathsf{H}\ \frac{E \xrightarrow{R} C}{E \xrightarrow{S} C} : R \sqsubseteq S \in \mathcal{O} \qquad \mathsf{R}_\circ\ \frac{E \xrightarrow{R_1} C \quad C \xrightarrow{R_2} D}{E \xrightarrow{S} D} : R_1 \circ R_2 \sqsubseteq S \in \mathcal{O}$$

Fig. 2 Basic inference rules with links for reasoning in $\mathcal{EL}_\bot^+$

$$
\begin{array}{lll}
C \sqcap D \sqsubseteq D & \text{by } \mathsf{R}_\sqcap^- \text{ to (3)} & (4) \\
A \sqsubseteq \exists R.D & \text{by } \mathsf{R}_\exists \text{ to (2) and (4)} & (5) \\
A \sqsubseteq \exists S.D & \text{by } \mathsf{R}_\mathsf{H} \text{ to (5) using (ax4)} & (6) \\
A \sqsubseteq A \sqcap \exists S.D & \text{by } \mathsf{R}_\sqcap^+ \text{ to (1) and (6)} & (7) \\
A \sqsubseteq B & \text{by } \mathsf{R}_\sqsubseteq \text{ to (7) using (ax2)} & (8)
\end{array}
$$

Note that (5) was produced by $\mathsf{R}_\exists$ using a derived premise (4). We would not be able to apply $\mathsf{R}_\exists$ to (2) if instead of this premise we had a side condition similarly to $R_\sqsubseteq$ because there are no (told) subsumptions in $\mathcal{O}$ with $C \sqcap D$ on the left-hand side.

Clearly, the rules in Fig. 1 can derive only subsumptions that are entailed by $\mathcal{O}$ since each rule produces a logical consequence of the premises and the side condition, if there is one. In other words, the inference system is *sound*. Furthermore, it is possible to show that all entailed subsumptions can be derived by the rules. That is, the rules are *complete*.

3.2 Redundant Rule Applications

We are not going to prove completeness of the rules in Fig. 1 just yet, but first present some optimizations. The main idea is that one can avoid applying some inferences to conclusions produced by $\mathsf{R}_\exists$ without loosing any derivable subsumptions. Specifically, such conclusions should not be used as the first premise of $\mathsf{R}_\bot$ and $\mathsf{R}_\exists$ and premises of R_H and $\mathsf{R}_\circ$—these are all places where existential restrictions are mentioned explicitly in the premises.

Example 2 Note that in Example 1 the conclusion (5) of the rule $\mathsf{R}_\exists$ was used as a premise of R_H to produce (6). Subsumption (6), however, can be produced differently by 'swapping' the order of application of $\mathsf{R}_\exists$ and R_H:

$$
\begin{array}{lll}
A \sqsubseteq \exists S.(C \sqcap D) & \text{by } \mathsf{R}_\mathsf{H} \text{ to (2) using (ax4)} & (9) \\
A \sqsubseteq \exists S.D & \text{by } \mathsf{R}_\exists \text{ to (9) and (4)} & (10)
\end{array}
$$

Hence, $A \sqsubseteq B$ can be derived without using conclusions of $\mathsf{R}_\exists$ as premises of R_H.

To formalize the described optimization, we first need to distinguish existential restrictions produced by $\mathsf{R}_\exists$ from those produced by other rules. For this purpose, we introduce another type of conclusions called *(existential) links* $C \xrightarrow{R} D$, semantically equivalent to $C \sqsubseteq \exists R.D$. The new rules using links are given in Fig. 2. Note that instead of a single rule $\mathsf{R}_\exists$ we now have two rules $\mathsf{R}_\exists^-$ and $\mathsf{R}_\exists^+$, a combined application of which gives $\mathsf{R}_\exists$. Intuitively, rule $\mathsf{R}_\exists^-$ eliminates existential restrictions by converting them to links, whereas rule $\mathsf{R}_\exists^+$ produces existential restrictions from links and other concept subsumptions.

Note that (so far) the inference rules in Fig. 2 derive $C \xrightarrow{R} D$ if and only if they derive $C \sqsubseteq \exists R.D$. Indeed, $C \xrightarrow{R} D$ can be derived from $C \sqsubseteq \exists R.D$ by $\mathsf{R}_\exists^-$. On the other hand, since $D \sqsubseteq D$ is derivable by R_0, we can derive $C \sqsubseteq \exists R.D$ from $C \xrightarrow{R} D$ by $\mathsf{R}_\exists^+$. It is easy to see that if we now replace every link in the rules of Fig. 2 by the corresponding existential restrictions, we obtain the rules in Fig. 1. Indeed, $\mathsf{R}_\exists^-$ will produce $E \sqsubseteq \exists R.C$ from $E \sqsubseteq \exists R.C$ and can be ignored, $\mathsf{R}_\exists^+$ in Fig. 2 becomes $\mathsf{R}_\exists$ in Fig. 1, whereas rules $\mathsf{R}_\bot$, R_H and $\mathsf{R}_\circ$ in Fig. 2 will become the corresponding rules in Fig. 1. Therefore, the rules in Fig. 2 derive the same concept subsumptions as those in Fig. 1. From now on, we only focus on the rules in Fig. 2.

Existential links in Fig. 2 prevent conclusions of $\mathsf{R}_\exists^+$ to be used as the first premise of $\mathsf{R}_\exists^+$ or as premises of R_H and $\mathsf{R}_\circ$ *immediately*, but, of course, one can first convert such conclusions to links using $\mathsf{R}_\exists^-$, after which they can be still used as premises of those rules. To avoid this situation, we are going to 'block' the applications of $\mathsf{R}_\exists^-$ to conclusions of $\mathsf{R}_\exists^+$.

Definition 1 (Redundancy of $\mathsf{R}_\exists^-$) Let $\mathcal{O}$ be an $\mathcal{EL}_\bot^+$ ontology, and $\mathsf{Closure}$ a set of subsumptions and existential links. An application of rule $\mathsf{R}_\exists^-$ that derives $E \xrightarrow{R} C$ from $E \sqsubseteq \exists R.C$ is *redundant w.r.t.* $\mathsf{Closure}$ if $\{E \xrightarrow{R} D, D \sqsubseteq C\} \subseteq \mathsf{Closure}$ for some concept D. Any other application of rules in Fig. 2 is *non-redundant w.r.t.* $\mathsf{Closure}$. If $\mathsf{Closure}$ is clear from the context, we often say that an inference is redundant or non-redundant without mentioning $\mathsf{Closure}$.

We say that $\mathsf{Closure}$ is *closed under the rules of Fig. 2 up to redundancy w.r.t.* $\mathcal{O}$ if it contains all conclusions of non-redundant applications of rules in Fig. 2 w.r.t. $\mathcal{O}$.

Example 3 Consider the ontology $\mathcal{O}$ from Example 1. We will use the inference rules from Fig. 2 to derive $A \sqsubseteq \exists S.D$. The shortest derivation proceeds as follows.

$A \sqsubseteq A$	by R_0	(11)
$A \sqsubseteq \exists R.(C \sqcap D)$	by $\mathsf{R}_\sqsubseteq$ to (11) using (ax1)	(12)
$A \xrightarrow{R} C \sqcap D$	by $\mathsf{R}_\exists^-$ to (12)	(13)
$C \sqcap D \sqsubseteq C \sqcap D$	by R_0	(14)
$C \sqcap D \sqsubseteq D$	by $\mathsf{R}_\sqcap^-$ to (14)	(15)
$A \xrightarrow{S} C \sqcap D$	by R_H to (13) using (ax4)	(16)
$A \sqsubseteq \exists S.D$	by $\mathsf{R}_\exists^+$ to (16) and (15)	(17)

There is also another derivation. It starts as above until (15), but then instead of deriving the link $A \xrightarrow{S} C \sqcap D$, it proceeds by using the link $A \xrightarrow{R} C \sqcap D$ as follows.

$A \sqsubseteq \exists R.D$	by $\mathsf{R}_\exists^+$ to (13) and (15)	(18)
$A \xrightarrow{R} D$	by $\mathsf{R}_\exists^-$ to (18)	(19)
$A \xrightarrow{S} D$	by R_H to (19) using (ax4)	(20)
$D \sqsubseteq D$	by R_0	(21)
$A \sqsubseteq \exists S.D$	by $\mathsf{R}_\exists^+$ to (20) and (21)	(22)

The redundancy condition allows us to 'block' the second derivation after (18): we have already derived $A \xrightarrow{R} C \sqcap D$ in (13) and $C \sqcap D \sqsubseteq D$ in (15), so the application of $\mathsf{R}_\exists^-$ in (19) is redundant. Intuitively, every further conclusion using the link $A \overset{R}{\to} D$ can as well be derived using the former link $A \overset{R}{\to} C \sqcap D$ instead.

Remark 1 Note that if $\mathsf{Closure}$ is closed under the rules in Fig. 2 up to redundancy, then every application of $\mathsf{R}^-_\exists$ to $E \sqsubseteq \exists R.C \in \mathsf{Closure}$ is redundant w.r.t. $\mathsf{Closure}$. Indeed, otherwise $\mathsf{Closure}$ contains the conclusion $E \overset{R}{\to} C$ of $\mathsf{R}^-_\exists$ by the definition of closure under redundancy, and since $C \sqsubseteq C \in \mathsf{Closure}$ due to closure under R_0, we have $\{E \overset{R}{\to} C, C \sqsubseteq C\} \subseteq \mathsf{Closure}$.

Note that the application of $\mathsf{R}^-_\exists$ to $E \sqsubseteq \exists R.C$ can be redundant even if $E \sqsubseteq \exists R.C$ was not obtained by rule $\mathsf{R}^+_\exists$, so the redundancy condition can also be used to avoid other inferences than those mentioned in the beginning of this section. In the next section, we prove that our optimized calculus is complete for deriving all subsumptions entailed in $\mathcal{EL}^+_\bot$.

Theorem 1 *Let $\mathcal{O}$ be an $\mathcal{EL}^+_\bot$ ontology, and let* $\mathsf{Closure}$ *be a set that is closed under the rules in Fig. 2 up to redundancy. Then $\mathcal{O} \models C \sqsubseteq D$ implies $C \sqsubseteq D \in \mathsf{Closure}$ or $C \sqsubseteq \bot \in \mathsf{Closure}$.*

From Theorem 1, it follows that if $\mathcal{O} \models C \sqsubseteq D$ then either $C \sqsubseteq D$ or $C \sqsubseteq \bot$ is derivable using the rules in Fig. 2 even if the applications of $\mathsf{R}^-_\exists$ are required to be non-redundant w.r.t. the set of already derived (intermediate) conclusions. Specifically, let $\{S_i \mid i \geq 0\}$ be a sequence of sets such that $S_0 = \emptyset$, and for every $i \geq 0$, let S_{i+1} be the extension of S_i with conclusions of all non-redundant (w.r.t. S_i) applications of rules from Fig. 2 to S_i. Let $\mathsf{Closure} = \bigcup_{i \geq 0} S_i$ be the set of all conclusions derivable in this way. Clearly, $\mathsf{Closure}$ is closed under the rules in Fig. 2 up to redundancy. Hence, by Theorem 1, if $\mathcal{O} \models C \sqsubseteq D$, then either $C \sqsubseteq D$ or $C \sqsubseteq \bot$ belongs to $\mathsf{Closure}$, i.e., is derivable using this particular strategy.

3.3 Proof of Completeness

The goal of this section is to prove Theorem 1. A commonly used technique for proving completeness of $\mathcal{EL}$ procedures is based on canonical model construction.[4] Similarly to saturation-based theorem proving [18], canonical models for $\mathcal{EL}$ are constructed (uniquely) from the set of expressions closed under inference rules (up to redundancy).

Assume that $\mathcal{O}$ and $\mathsf{Closure}$ satisfy the condition of Theorem 1. If $C \sqsubseteq \bot \in \mathsf{Closure}$ for every concept C then Theorem 1 trivially holds. So, w.l.o.g., we can assume that $C \sqsubseteq \bot \notin \mathsf{Closure}$ for at least one concept C. In this case, one can define the canonical model (of $\mathcal{O}$ w.r.t. $\mathsf{Closure}$) as follows:

Definition 2 (Canonical Model) The *canonical model* $\mathcal{I}$ is defined by

$$\begin{aligned}
\Delta^\mathcal{I} &= \{x_C \mid C \sqsubseteq \bot \notin \mathsf{Closure}\},\\
A^\mathcal{I} &= \{x_C \in \Delta^\mathcal{I} \mid C \sqsubseteq A \in \mathsf{Closure}\},\\
R^\mathcal{I} &= \{\langle x_C, x_D\rangle \in \Delta^\mathcal{I} \times \Delta^\mathcal{I} \mid C \overset{R}{\to} D \in \mathsf{Closure}\}.
\end{aligned}$$

Note that $\mathcal{I}$ is well-defined only if $C \sqsubseteq \bot \notin \mathsf{Closure}$ for at least one concept C. Otherwise, the domain $\Delta^\mathcal{I}$ is empty and $\mathcal{I}$ is not a valid interpretation. Note also that the definition of the canonical model uses only expressions of the form $C \sqsubseteq A$ and $C \overset{R}{\to} D$ from $\mathsf{Closure}$. In particular, the existential restrictions $C \sqsubseteq \exists R.D$ are ignored by this definition, in contrast to the corresponding links $C \overset{R}{\to} D$, which define the interpretation of roles.

[4] In the earlier $\mathcal{EL}$ papers [6,8,9], canonical models were called *completion graphs*.

Of course, even if $\mathcal{I}$ is well-defined, it is not necessarily a model of $\mathcal{O}$. In the rest of this section we demonstrate that $\mathcal{I} \models \mathcal{O}$ if $\mathsf{Closure}$ satisfies the closure properties of Theorem 1. Towards this goal, we first prove two auxiliary lemmas.

Lemma 1 *For each $x_C \in \Delta^{\mathcal{I}}$ and each concept D, $C \sqsubseteq D \in \mathsf{Closure}$ implies $x_C \in D^{\mathcal{I}}$.*

Proof The proof is by induction over the structure of D. In each case, we assume that $x_C \in \Delta^{\mathcal{I}}$ and $C \sqsubseteq D \in \mathsf{Closure}$, and we prove that $x_C \in D^{\mathcal{I}}$.

- Case $D = A$: $x_C \in A^{\mathcal{I}}$ holds by the definition of $A^{\mathcal{I}}$.
- Case $D = \top$: We have $x_C \in \top^{\mathcal{I}}$ since $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$.
- Case $D = \bot$: Not possible since, by Definition 2, $C \sqsubseteq \bot \notin \mathsf{Closure}$ for $x_C \in \Delta^{\mathcal{I}}$.
- Case $D = D_1 \sqcap D_2$: Due to closure under $\mathsf{R}^-_\sqcap$, $C \sqsubseteq D_1 \sqcap D_2 \in \mathsf{Closure}$ implies $C \sqsubseteq D_i \in \mathsf{Closure}$ for $i = 1,2$. Then, by the induction hypothesis, we have $x_C \in D_i^{\mathcal{I}}$, from which $x_C \in (D_1 \sqcap D_2)^{\mathcal{I}}$ follows by the semantics of $\sqcap$.
- Case $D = \exists R.D_2$: Due to closure under $\mathsf{R}^-_\exists$ up to redundancy, by Remark 1, there exists D_1 such that $\{C \overset{R}{\to} D_1, D_1 \sqsubseteq D_2\} \subseteq \mathsf{Closure}$. Now if $D_1 \sqsubseteq \bot \in \mathsf{Closure}$ then $C \sqsubseteq \bot \in \mathsf{Closure}$ due to closure under $\mathsf{R}_\bot$, which contradicts $x_C \in \Delta^{\mathcal{I}}$. Therefore, $D_1 \sqsubseteq \bot \notin \mathsf{Closure}$, and so $x_{D_1} \in \Delta^{\mathcal{I}}$. Since $D_1 \sqsubseteq D_2 \in \mathsf{Closure}$, by induction hypothesis applied to $x_{D_1} \in \Delta^{\mathcal{I}}$, we get $x_{D_1} \in D_2^{\mathcal{I}}$. Since $C \overset{R}{\to} D_1 \in \mathsf{Closure}$, we have $\langle x_C, x_{D_1}\rangle \in R^{\mathcal{I}}$ by the definition of $R^{\mathcal{I}}$. Then $x_C \in (\exists R.D_2)^{\mathcal{I}}$ follows by the semantics of $\exists$. □

Corollary 1 *For each C such that $C \sqsubseteq \bot \notin \mathsf{Closure}$ we have $x_C \in C^{\mathcal{I}}$.*

Proof This is a direct application of Lemma 1 with $C = D$; the required $C \sqsubseteq C \in \mathsf{Closure}$ holds due to closure under R_0.

Note that the proof of Lemma 1 uses only that $\mathsf{Closure}$ is closed under $\mathsf{R}^-_\sqcap$, $\mathsf{R}_\bot$, R_0, and under $\mathsf{R}^-_\exists$ up to redundancy. The converse of Lemma 1 uses closure under $\mathsf{R}_\top$, $\mathsf{R}^+_\sqcap$ and $\mathsf{R}^+_\exists$:

Lemma 2 *For each $x_C \in \Delta^{\mathcal{I}}$ and each concept D, $x_C \in D^{\mathcal{I}}$ implies $C \sqsubseteq D \in \mathsf{Closure}$.*

Proof The proof is by induction over the structure of D. In each case, we assume that $x_C \in D^{\mathcal{I}}$, and we prove that $C \sqsubseteq D \in \mathsf{Closure}$.

- Case $D = A$: $C \sqsubseteq A \in \mathsf{Closure}$ holds by the definition of $A^{\mathcal{I}}$.
- Case $D = \top$: $C \sqsubseteq \top \in \mathsf{Closure}$ holds due to closure under $\mathsf{R}_\top$.
- Case $D = \bot$: This case cannot occur since $x_C \in \bot^{\mathcal{I}} = \emptyset$ is not possible.
- Case $D = D_1 \sqcap D_2$: By the semantics of $\sqcap$, $x_C \in (D_1 \sqcap D_2)^{\mathcal{I}}$ implies $x_C \in D_i^{\mathcal{I}}$ for $i = 1,2$. Then, by the induction hypothesis, we have $C \sqsubseteq D_i \in \mathsf{Closure}$ for $i = 1,2$, from which $C \sqsubseteq D_1 \sqcap D_2 \in \mathsf{Closure}$ follows due to closure under $\mathsf{R}^+_\sqcap$.
- Case $D = \exists R.D_2$: By the semantics of $\exists$, $x_C \in (\exists R.D_2)^{\mathcal{I}}$ implies that there exists an element $x_{D_1} \in \Delta^{\mathcal{I}}$ such that $\langle x_C, x_{D_1}\rangle \in R^{\mathcal{I}}$ and $x_{D_1} \in D_2^{\mathcal{I}}$. Then $C \overset{R}{\to} D_1 \in \mathsf{Closure}$ by the definition of $R^{\mathcal{I}}$ and, by applying the induction hypothesis to $x_{D_1} \in D_2^{\mathcal{I}}$, we obtain $D_1 \sqsubseteq D_2 \in \mathsf{Closure}$. Then $C \sqsubseteq \exists R.D_2 \in \mathsf{Closure}$ follows due to closure under $\mathsf{R}^+_\exists$. □

We are now ready to prove that $\mathcal{I}$ is indeed a model of $\mathcal{O}$.

Theorem 2 *$\mathcal{I} \models \mathcal{O}$.*

Proof We check that $\mathcal{I} \models \alpha$ for each axiom $\alpha \in \mathcal{O}$:

– Case $\alpha = D \sqsubseteq E$: We consider an arbitrary $x_C \in D^{\mathcal{I}}$. We need to show that $x_C \in E^{\mathcal{I}}$. Indeed, by Lemma 2, $x_C \in D^{\mathcal{I}}$ implies $C \sqsubseteq D \in \mathsf{Closure}$. Then $C \sqsubseteq E \in \mathsf{Closure}$ due to closure under $\mathsf{R}_{\sqsubseteq}$, from which $x_C \in E^{\mathcal{I}}$ follows by Lemma 1.
– Case $\alpha = R \sqsubseteq S$: We consider arbitrary $\langle x_C, x_D \rangle \in R^{\mathcal{I}}$. We need to show that $\langle x_C, x_D \rangle \in S^{\mathcal{I}}$. Indeed, by the definition of $R^{\mathcal{I}}$, we have $C \overset{R}{\to} D \in \mathsf{Closure}$. Then $C \overset{S}{\to} D \in \mathsf{Closure}$ due to closure under $\mathsf{R_H}$, from which $\langle x_C, x_D \rangle \in S^{\mathcal{I}}$ follows by the definition of $S^{\mathcal{I}}$.
– Case $\alpha = R_1 \circ R_2 \sqsubseteq S$: We consider arbitrary $\langle x_C, x_D \rangle \in R_1^{\mathcal{I}}$ and $\langle x_D, x_E \rangle \in R_2^{\mathcal{I}}$. We need to show that $\langle x_C, x_E \rangle \in S^{\mathcal{I}}$. Indeed, by the definition of $R_1^{\mathcal{I}}$ and $R_2^{\mathcal{I}}$, we have $C \overset{R_1}{\to} D \in \mathsf{Closure}$ and $D \overset{R_2}{\to} E \in \mathsf{Closure}$. Then $C \overset{S}{\to} E \in \mathsf{Closure}$ due closure under $\mathsf{R}_{\circ}$, from which $\langle x_C, x_E \rangle \in S^{\mathcal{I}}$ follows by the definition of $S^{\mathcal{I}}$. □

It is now easy to conclude the proof of Theorem 1. Let C and D be arbitrary concepts such that $\mathcal{O} \models C \sqsubseteq D$ and $C \sqsubseteq \bot \notin \mathsf{Closure}$. Then, the canonical model $\mathcal{I}$ of $\mathcal{O}$ is well-defined, and by Theorem 2, $\mathcal{I} \models \mathcal{O}$. Since $C \sqsubseteq \bot \notin \mathsf{Closure}$, $x_C \in \Delta^{\mathcal{I}}$, and by Corollary 1, $x_C \in C^{\mathcal{I}}$. Since $\mathcal{I} \models \mathcal{O} \models C \sqsubseteq D$, we have $x_C \in D^{\mathcal{I}}$, hence $C \sqsubseteq D \in \mathsf{Closure}$ by Lemma 2.

3.4 Goal-Directed Rule Application

The rules in Fig. 2 can derive an infinite number of subsumptions since already rule $\mathsf{R_0}$ can be applied for any $\mathcal{EL}_{\bot}^{+}$ concept C. In practice, however, one is usually interested in finitely many subsumptions of some specific form. For example, for classification, it is sufficient to derive only those $C \sqsubseteq D$ where C and D are either atomic concepts, $\top$, or $\bot$. Deriving subsumptions from a given finite set, however, may require other subsumptions not in this set to be derived first, and in general, it is not clear why only finitely many intermediate conclusions can be considered. In this section, we present restrictions of the inference rules in Fig. 2 that can be used to avoid many inferences that are not relevant for deriving the goal subsumptions. To this end, within this section we assume that we are given a fixed $\mathcal{EL}_{\bot}^{+}$ ontology $\mathcal{O}$ and a fixed goal subsumption $F \sqsubseteq G$. Our goal is to determine which applications of the rules in Fig. 2 are needed for deriving this goal subsumption, if it is derivable.

First we demonstrate that it is sufficient to consider only intermediate conclusions that are reachable from the goal using derivable links.

Definition 3 Given a concept C and a set $\mathsf{Closure}$, we say that C is *reachable from F in* $\mathsf{Closure}$ if there exist sequences of concepts $F = C_0, C_1, \ldots, C_n = C$ and roles $R_1, \ldots, R_n$, $n \geq 0$, such that $C_{i-1} \overset{R_i}{\to} C_i \in \mathsf{Closure}$ for every i with $1 \leq i \leq n$. We say that expressions $C \sqsubseteq D$ and $C \overset{R}{\to} D$ are *reachable from F in* $\mathsf{Closure}$ if C is reachable from F in $\mathsf{Closure}$.

Note that an expression can be reachable from F in $\mathsf{Closure}$ even if it is not contained in $\mathsf{Closure}$. When speaking about reachable concepts and expressions, we often drop F and $\mathsf{Closure}$ if they are clear from the context. The following result demonstrates that non-reachable conclusions can be always disregarded when deriving the goal subsumption.

Lemma 3 *Let* $\mathsf{Closure}$ *be a set that contains all reachable (from F in* $\mathsf{Closure}$*) conclusions of non-redundant applications of rules in Fig. 2 to* $\mathsf{Closure}$*. Then $\mathcal{O} \models F \sqsubseteq G$ implies that either $F \sqsubseteq G \in$* $\mathsf{Closure}$ *or $F \sqsubseteq \bot \in$* $\mathsf{Closure}$.

Proof Let $\{S_i \mid i \geq 0\}$ be a sequence of sets of expressions such that $S_0 = \mathsf{Closure}$, and for every $i \geq 0$, S_{i+1} is the extension of S_i with all conclusions of non-redundant applications of rules to S_i. Let $S = \bigcup_{i \geq 0} S_i$. Clearly, S is closed under all rules in Fig. 2 up to redundancy. Thus, by Theorem 1, if $\mathcal{O} \models F \sqsubseteq G$, then S contains either $F \sqsubseteq \bot$ or $F \sqsubseteq G$.

We will now prove by induction on $i \geq 0$ that if $\alpha \in S_i$ is reachable from F in $\mathsf{Closure}$, then $\alpha \in \mathsf{Closure}$. Since both $F \sqsubseteq \bot$ and $F \sqsubseteq G$ are reachable from F in $\mathsf{Closure}$, this will imply the claim of the lemma.

The induction hypothesis clearly holds for $i = 0$ since $S_0 = \mathsf{Closure}$ by definition. For the induction step, assume that the hypothesis holds for some $i \geq 0$. Now take any $\alpha \in S_{i+1}$ that is reachable from F in $\mathsf{Closure}$. If $\alpha \in S_i$ then $\alpha \in \mathsf{Closure}$ by induction hypothesis. Otherwise, $\alpha \in S_{i+1} \setminus S_i$ is a conclusion of a non-redundant application of some rule to S_i. We prove that $\alpha \in \mathsf{Closure}$ by considering all possible cases for deriving α:

- rules R_0, $\mathsf{R}_\top$, $\mathsf{R}_\sqcap^-$, $\mathsf{R}_\sqcap^+$, $\mathsf{R}_\exists^-$, $\mathsf{R}_\sqcap$, R_H: it is easy to see that all premises of such rules are reachable if α is reachable (since they have the same left-hand side). Since all premises belong to S_i and are reachable, by induction hypothesis, they belong to $\mathsf{Closure}$. Since $\mathsf{Closure}$ contains all reachable conclusions of non-redundant rule applications, we obtain that $\alpha \in \mathsf{Closure}$ (note that if an application of $\mathsf{R}_\exists^-$ is not redundant w.r.t. S_i then it is not redundant w.r.t. $\mathsf{Closure}$ because $\mathsf{Closure} \subseteq S_i$).
- rules $\mathsf{R}_\bot$, $\mathsf{R}_\exists^+$, $\mathsf{R}_\circ$: for these rules, the first premise is reachable if the conclusion is reachable. Hence the first premise belongs to $\mathsf{Closure}$ by induction hypothesis. But then, the second premise is also reachable by Definition 3, and likewise belongs to $\mathsf{Closure}$. Since both premises belong to $\mathsf{Closure}$, we obtain similarly that $\alpha \in \mathsf{Closure}$. □

It is possible to restrict the application of inference rules even further. Specifically, the inference rules $\mathsf{R}_\top$, $\mathsf{R}_\sqcap^+$, $\mathsf{R}_\exists^+$ that introduce new concepts on the right-hand side of subsumptions can be restricted to produce only concepts occurring in G or occurring in the left-hand sides of axioms in $\mathcal{O}$.

Definition 4 We say that a concept C *occurs negatively* (respectively *positively*) in a ontology $\mathcal{O}$, if C is a syntactic subexpression of D (respectively E) for some $D \sqsubseteq E \in \mathcal{O}$. An application of rules $\mathsf{R}_\top$, $\mathsf{R}_\sqcap^+$, or $\mathsf{R}_\exists^+$ is *goal-directed* (w.r.t. $\mathcal{O}$ and $F \sqsubseteq G$) if for its conclusion $C \sqsubseteq D$, either D occurs in G or occurs negatively in $\mathcal{O}$. Applications of rules other than $\mathsf{R}_\top$, $\mathsf{R}_\sqcap^+$, and $\mathsf{R}_\exists^+$ (that do not introduce new concepts) are always goal-directed.

Lemma 4 *Let* $\mathsf{Closure}$ *be a set that contains all reachable (from F in* $\mathsf{Closure}$*) conclusions of non-redundant, goal-directed applications of rules in Fig. 2. Then* $\mathcal{O} \models F \sqsubseteq G$ *implies that either* $F \sqsubseteq G \in \mathsf{Closure}$ *or* $F \sqsubseteq \bot \in \mathsf{Closure}$.

Proof Let $\{S_i \mid i \geq 0\}$ be a sequence of sets of expressions such that $S_0 = \mathsf{Closure}$, and for every $i \geq 0$, S_{i+1} is the extension of S_i with all reachable (from F in $\mathsf{Closure}$) conclusions of *not* goal-directed applications of rules to S_i, i.e., those conclusions $C \sqsubseteq D$ of $\mathsf{R}_\top$, $\mathsf{R}_\sqcap^+$, $\mathsf{R}_\exists^+$ such that D occurs neither in G, nor negatively in $\mathcal{O}$.

First, we prove by induction on $i \geq 0$ that each S_{i+1} contains all reachable conclusions of non-redundant rule applications to S_i. Indeed, for $i = 0$ this is the case since, by the condition of the lemma, $S_0 = \mathsf{Closure}$ contains all such conclusions for goal-directed rule applications, and S_1 contains all such conclusions for not goal-directed rule applications. Now suppose that this property holds for some $i \geq 0$, and let α be a reachable conclusion of some non-redundant rule application to S_{i+1}. We need to prove that $\alpha \in S_{i+2}$. If α was derived from premises in S_i, then $\alpha \in S_{i+1}$ by induction hypothesis, hence $\alpha \in S_{i+2}$. Otherwise, α was derived from some $\beta = C \sqsubseteq D \in S_{i+1} \setminus S_i$, which is obtained by one of the rules $\mathsf{R}_\top$, $\mathsf{R}_\sqcap^+$, or $\mathsf{R}_\exists^+$ such that D occurs neither in G nor negatively in $\mathcal{O}$. But then α can be obtained from β only using $\mathsf{R}_\sqcap^-$, $\mathsf{R}_\sqcap^+$, $\mathsf{R}_\exists^-$, or $\mathsf{R}_\exists^+$ (the premises of other rules cannot have such β). Then:

$$\mathsf{R}_0\ \frac{\mathsf{init}(C)}{C \sqsubseteq C} \qquad \mathsf{R}_\top\ \frac{\mathsf{init}(C)}{C \sqsubseteq \top} : \top \text{ occurs negatively in } \mathcal{O} \qquad \mathsf{R}_\bot\ \frac{E \overset{R}{\to} C \quad C \sqsubseteq \bot}{E \sqsubseteq \bot}$$

$$\mathsf{R}_\sqcap^-\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2} \qquad \mathsf{R}_\sqcap^+\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occur negatively in } \mathcal{O}$$

$$\mathsf{R}_\exists^-\ \frac{E \sqsubseteq \exists R.C}{E \overset{R}{\to} C} \qquad \mathsf{R}_\exists^+\ \frac{E \overset{R}{\to} C \quad C \sqsubseteq D}{E \sqsubseteq \exists S.D} : \begin{array}{l} R \sqsubseteq_\mathcal{O}^* S \\ \exists S.D \text{ occurs negatively in } \mathcal{O} \end{array}$$

$$\mathsf{R}_\sqsubseteq\ \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O} \qquad \mathsf{R}_\circ\ \frac{E \overset{R_1}{\to} C \quad C \overset{R_2}{\to} D}{E \overset{S}{\to} D} : \begin{array}{l} R_1 \sqsubseteq_\mathcal{O}^* S_1 \\ R_2 \sqsubseteq_\mathcal{O}^* S_2 \\ S_1 \circ S_2 \sqsubseteq S \in \mathcal{O} \end{array} \qquad \mathsf{R}_\rightsquigarrow\ \frac{E \overset{R}{\to} C}{\mathsf{init}(C)}$$

Fig. 3 Optimized inference rules for classification of $\mathcal{EL}_\bot^+$ ontologies

- if α is obtained by $\mathsf{R}_\sqcap^-$, then $\beta = C \sqsubseteq D_1 \sqcap D_2$ can only be obtained by $R_\sqcap^+$ from S_i, and so $\alpha \in S_i \subseteq S_{i+1} \subseteq S_{i+2}$.
- if α is obtained by $\mathsf{R}_\exists^-$, then $\beta = E \sqsubseteq \exists R.C$ can only be obtained by $R_\exists^+$ from S_i, so this rule application is redundant and cannot take place.
- if $\alpha = C \sqsubseteq D_1 \sqcap D_2$ is obtained by $\mathsf{R}_\sqcap^+$, then $\beta = C \sqsubseteq D_1$ or $\beta = C \sqsubseteq D_2$, and so, respectively, either D_1 or D_2 occurs neither in G nor negatively in $\mathcal{O}$. But then the same holds for $D_1 \sqcap D_2$, and so $\alpha \in S_{i+2}$ by the definition of S_{i+2}.
- if $\alpha = \exists R.D$ is obtained by $\mathsf{R}_\exists^+$, then $\beta = C \sqsubseteq D$ and D occurs neither in F nor negatively in $\mathcal{O}$. But then the same holds for $\exists R.D$, and so $\alpha \in S_{i+2}$ by the definition of S_{i+2}.

Now since each S_{i+1} contains all reachable conclusions of non-redundant applications of rules to S_i, the set $S = \bigcup_{i \geq 0} S_i$ is closed under such inferences. Thus, by Lemma 3, if $\mathcal{O} \models F \sqsubseteq G$ then S contains either $F \sqsubseteq G$ or $F \sqsubseteq \bot$. Since neither $F \sqsubseteq G$ nor $F \sqsubseteq \bot$ can be obtained by inferences producing S_{i+1} from S_i for $i \geq 0$ (for $F \sqsubseteq G$ this is not possible since G occurs in G, and $F \sqsubseteq \bot$ can be obtained by neither of $\mathsf{R}_\top$, $\mathsf{R}_\sqcap^+$ or $\mathsf{R}_\exists^+$), $S_0 = \mathsf{Closure}$ must contain either $F \sqsubseteq G$ or $F \sqsubseteq \bot$, as required. □

3.5 Optimized Inference Rules for Classification

In this section we formulate an optimized calculus for computing classification of $\mathcal{EL}_\bot^+$ ontologies that takes into account the optimizations described in the previous section. Recall that for computing classification it is sufficient to derive all subsumptions of the form $A \sqsubseteq B$, $A \sqsubseteq \bot$, $\top \sqsubseteq B$, or $\top \sqsubseteq \bot$ where A and B are atomic concepts. By Lemma 4, it is thus sufficient to restrict the rules $\mathsf{R}_\top$, $\mathsf{R}_\sqcap^+$, $\mathsf{R}_\exists^+$ in Fig. 2 to produce only subsumptions $C \sqsubseteq D$ where D occurs negatively in $\mathcal{O}$: note that for these rules, D cannot be an atomic concept or $\bot$. To make sure that only reachable conclusions are derived by the rules, we use a special expression $\mathsf{init}(C)$ to indicate that the C is *initialized* as a reachable concept. This expression is then used as a premise of the rules R_0 and $\mathsf{R}_\top$ to make sure that the conclusions produced by such rules are reachable. We do not need to add similar premises to other rules since the conclusions of these rules have the same left-hand side as one of the premises of the rules, and hence are reachable whenever the premises are. Whenever we derive a (reachable) link $E \overset{R}{\to} C$ for some new C, we need to produce $\mathsf{init}(C)$ to indicate that C is reachable as well; to this end, we introduce a special rule $\mathsf{R}_\rightsquigarrow$ that implements this inference.

The optimized inference system is presented in Fig. 3. Note that we have also removed the rule R_H and, instead, use new side conditions of the form $R \sqsubseteq_\mathcal{O}^* S$ for the rules $\mathsf{R}_\exists^+$ and $\mathsf{R}_\circ$.

Here $\sqsubseteq^*_{\mathcal{O}}$ is the transitive closure of the role hierarchy in $\mathcal{O}$, i.e., it is the smallest reflexive transitive binary relation over roles such that $R \sqsubseteq^*_{\mathcal{O}} S$ holds for all $R \sqsubseteq S \in \mathcal{O}$. It is easy to see that once the rules $\mathsf{R}^+_\exists$ and $\mathsf{R}_\circ$ are relaxed in this way, the rule $\mathsf{R_H}$ is not needed anymore: if the conclusion of this rule can be used in $\mathsf{R}^+_\exists$ or $\mathsf{R}_\circ$, then its premise can be used instead to produce the same conclusion. The advantage of the new rules is that they produce fewer links: note that if the rules in Fig. 2 derive $E \xrightarrow{R} C$ then they also derive $E \xrightarrow{S} C$ for all S with $R \sqsubseteq^*_{\mathcal{O}} S$, which is not the case for the rules in Fig. 3. Thus, by precomputing the (often relatively small) role hierarchy $\sqsubseteq^*_{\mathcal{O}}$, we can reduce the number of derived links $E \xrightarrow{R} C$ by a factor of, roughly, the average number of superroles for each role in the role hierarchy. We also can generalize the notion of redundancy to the new rule $\mathsf{R}^-_\exists$ as follows:

Definition 5 (Redundancy of $\mathsf{R}^-_\exists$ for the system in Fig. 3) Let $\mathcal{O}$ be an $\mathcal{EL}^+_\bot$ ontology, and $\mathsf{Closure}$ a set of subsumptions, existential links, and initializations. An application of rule $\mathsf{R}^-_\exists$ that derives $E \xrightarrow{R} C$ from $E \sqsubseteq \exists R.C$ is *redundant w.r.t.* $\mathsf{Closure}$ *and* $\mathcal{O}$ if $\{E \xrightarrow{S} D, D \sqsubseteq C\} \subseteq \mathsf{Closure}$ for some concept D and some role $S \sqsubseteq^*_{\mathcal{O}} R$. Any other application of rules in Fig. 3 is *non-redundant w.r.t.* $\mathsf{Closure}$. We omit $\mathsf{Closure}$ if it is clear from the context.

$\mathsf{Closure}$ is *closed under the rules in Fig. 3 up to redundancy w.r.t.* $\mathcal{O}$ if it contains all conclusions of non-redundant applications of rules in Fig. 3 w.r.t. $\mathcal{O}$.

Now the analogue of Theorem 1 for the rules in Fig. 3 can be formulated as follows:

Theorem 3 *Let $\mathcal{O}$ be an $\mathcal{EL}^+_\bot$ ontology, and let* $\mathsf{Closure}$ *be a set closed under the rules in Fig. 3 up to redundancy w.r.t. $\mathcal{O}$. Then for every concept C such that* $\mathsf{init}(C) \in \mathsf{Closure}$ *and every D an atomic concept or $\bot$, $\mathcal{O} \models C \sqsubseteq D$ implies $C \sqsubseteq \bot \in$* $\mathsf{Closure}$ *or $C \sqsubseteq D \in$* $\mathsf{Closure}$.

Proof Let $\mathsf{Closure}'$ be obtained from $\mathsf{Closure}$ by removing all expressions of the form $\mathsf{init}(C)$ and adding $C \xrightarrow{S} D$ for every $C \xrightarrow{R} D \in \mathsf{Closure}$ with $R \sqsubseteq^*_{\mathcal{O}} S$. We will prove that $\mathsf{Closure}'$ satisfies the condition of Lemma 4 for every $F \sqsubseteq G$ such that $\mathsf{init}(F) \in \mathsf{Closure}$ and G is either an atomic concept or $\bot$. This will imply the claim of the theorem.

Since $\mathsf{Closure}$ is closed under the rules in Fig. 3 up to redundancy, $\mathsf{Closure}'$ is also closed under these rules up to redundancy. But then $\mathsf{Closure}'$ is closed under all applications of $\mathsf{R}^-_\sqcap$, $\mathsf{R}_\sqsubseteq$, $\mathsf{R_H}$, and $\mathsf{R}_\circ$ in Fig. 2 and under all goal-directed applications of rules $\mathsf{R}_\bot$, $\mathsf{R}^+_\sqcap$, and $\mathsf{R}^+_\exists$ in Fig. 2. Indeed, it is immediate that the additional side-conditions of $\mathsf{R}_\bot$, $\mathsf{R}^+_\sqcap$, and $\mathsf{R}^+_\exists$ in Fig. 3 exclude only rule applications that are not goal-directed.

Furthermore, $\mathsf{Closure}'$ is closed under all non-redundant applications of rule $\mathsf{R}^-_\exists$ in Fig. 2. Indeed, if an application of $R^-_\exists$ to $E \sqsubseteq \exists R.C$ is redundant w.r.t. $\mathsf{Closure}$ and $\mathcal{O}$ in the sense of Definition 5, then $\{E \xrightarrow{S} D, D \sqsubseteq C\} \subseteq \mathsf{Closure}$ for some D and some $S \sqsubseteq^*_{\mathcal{O}} R$. Hence $\{E \xrightarrow{R} D, D \sqsubseteq C\} \subseteq \mathsf{Closure}'$, and thus the application of $R^-_\exists$ to $E \sqsubseteq \exists R.C$ is redundant w.r.t. $\mathsf{Closure}'$ in the sense of Definition 1.

It remains thus to show that $\mathsf{Closure}'$ is closed under applications of $\mathsf{R_0}$ and $\mathsf{R}_\top$ in Fig. 2 that produce reachable conclusions. Indeed, since $\mathsf{init}(F) \in \mathsf{Closure}$ and $\mathsf{Closure}$ is closed under rule $\mathsf{R}_\rightsquigarrow$, we have $\mathsf{init}(C) \in \mathsf{Closure}$ for every C reachable from F in $\mathsf{Closure}$. Hence, every reachable conclusion of rules $\mathsf{R_0}$ and $\mathsf{R}_\top$ in Fig. 2 is in $\mathsf{Closure}'$ since $\mathsf{init}(C) \in \mathsf{Closure}$ for such conclusions, and $\mathsf{Closure}$ is closed under $\mathsf{R_0}$ and $\mathsf{R}_\top$ in Fig. 3. □

Example 4 Consider the ontology $\mathcal{O}$ from Example 1. To compute all subsumers of A, we compute the closure of $\mathsf{Input} = \{\mathsf{init}(A)\}$ under the inference rules in Fig. 3.

$\mathsf{init}(A)$	input expression	(23)
$A \sqsubseteq A$	by $\mathsf{R_0}$ to (23)	(24)

$$A \sqsubseteq \exists R.(C \sqcap D) \qquad \text{by } \mathsf{R}_{\sqsubseteq} \text{ to (24) using (ax1)} \tag{25}$$

$$A \overset{R}{\to} C \sqcap D \qquad \text{by } \mathsf{R}^{-}_{\exists} \text{ to (25)} \tag{26}$$

$$\mathsf{init}(C \sqcap D) \qquad \text{by } \mathsf{R}_{\rightsquigarrow} \text{ to (26)} \tag{27}$$

$$C \sqcap D \sqsubseteq C \sqcap D \qquad \text{by } \mathsf{R}_{0} \text{ to (27)} \tag{28}$$

$$C \sqcap D \sqsubseteq C \qquad \text{by } \mathsf{R}^{-}_{\sqcap} \text{ to (28)} \tag{29}$$

$$C \sqcap D \sqsubseteq D \qquad \text{by } \mathsf{R}^{-}_{\sqcap} \text{ to (28)} \tag{30}$$

$$A \sqsubseteq \exists S.D \qquad \text{by } \mathsf{R}^{+}_{\exists} \text{ to (26) and (30) using (ax4)} \tag{31}$$

$$A \sqsubseteq A \sqcap \exists S.D \qquad \text{by } \mathsf{R}^{+}_{\sqcap} \text{ to (24) and (31)} \tag{32}$$

$$A \overset{S}{\to} D \qquad \text{by } \mathsf{R}^{-}_{\exists} \text{ to (31)} \tag{33}$$

$$A \sqsubseteq C \qquad \text{by } \mathsf{R}_{\sqsubseteq} \text{ to (31) using (ax3)} \tag{34}$$

$$A \sqsubseteq B \qquad \text{by } \mathsf{R}_{\sqsubseteq} \text{ to (32) using (ax2)} \tag{35}$$

$$\mathsf{init}(D) \qquad \text{by } \mathsf{R}_{\rightsquigarrow} \text{ to (33)} \tag{36}$$

$$D \sqsubseteq D \qquad \text{by } \mathsf{R}_{0} \text{ to (36)} \tag{37}$$

The applications of rules $\mathsf{R}^{+}_{\exists}$ and $\mathsf{R}^{+}_{\sqcap}$ producing (31) and (32) use the fact that the concepts $\exists S.D$ and $A \sqcap \exists S.D$ occur negatively in $A \sqcap \exists S.D \sqsubseteq B$, which is a part of (ax2). Intuitively, these rules are used to gradually construct the subsumption $A \sqsubseteq A \sqcap \exists S.C$, so that rule $\mathsf{R}_{\sqsubseteq}$ with side condition (ax2) can be applied to derive $A \sqsubseteq B$.

Since the subsumption (31) was produced by rule $\mathsf{R}^{+}_{\exists}$, the application of $\mathsf{R}^{-}_{\exists}$ producing (33) is redundant. In fact, the expressions $\{(23)–(32), (34)–(35)\}$ are already closed under the rules in Fig. 3 up to redundancy. Note that rule $\mathsf{R}_{\bot}$ is not applicable, since $\bot$ does not occur in the ontology.

We can see that the following subsumers of A have been derived: A, $\exists R.(C \sqcap D)$, $\exists S.D$, $A \sqcap \exists S.D$, C, and B. By applying Theorem 3, we can also conclude that $\mathcal{O} \not\models A \sqsubseteq \bot$ and $\mathcal{O} \not\models A \sqsubseteq D$, but we cannot tell, e.g., whether $\mathcal{O} \models A \sqsubseteq \exists R.B$.

Finally, we estimate the worst-case complexity of computing the closure of Input under the inference rules in Fig. 3 w.r.t. $\mathcal{O}$. The key insight is to note that the rules in Fig. 3 can only produce expressions $\mathsf{init}(C)$, $C \sqsubseteq D$, and $C \overset{R}{\to} D$ such that all C, D, and R occur in Input or in $\mathcal{O}$. Let c and r be respectively the number of such concepts and roles. Thus, the inference rules can derive at most c expressions $\mathsf{init}(C)$, at most c^2 expressions $C \sqsubseteq D$, and at most $c^2 r$ expressions $C \overset{R}{\to} D$. In addition, we need to precompute at most r^2 subrole relations $R \sqsubseteq^*_{\mathcal{O}} S$. Consequently, the space complexity of the procedure is $O(c^2 r + r^2)$. To bound the time complexity, it is sufficient to estimate the number of instances of the inference rules in Fig. 3; the closure of a system of instantiated rules can be computed in time linear in the number of rules using the linear time algorithm for checking satisfiability of propositional Horn logic [30]. There are at most $c^3 r^3$ instances of rule $\mathsf{R}_{\circ}$, and this clearly dominates the remaining rules. Consequently, the time complexity of the procedure is $O(c^3 r^3)$. Thus, computing the closure of Input under the inference rules in Fig. 3 w.r.t. $\mathcal{O}$ can be implemented in polynomial time in the size of Input and $\mathcal{O}$.

3.6 Relations with Other Rule Systems

The inference rules in Fig. 2 and 3 are closely related to the *completion rules* proposed for $\mathcal{EL}^{++}$ [8]. The differences are mostly presentational. Instead of working with expressions of the form $C \sqsubseteq D$ and $C \overset{R}{\to} D$, the completion rules work with collection of sets of

CR1	If $D \in \mathbf{S}(C)$, $D \sqsubseteq E \in \mathcal{O}$, and $E \notin \mathbf{S}(C)$, then add E to $\mathbf{S}(C)$.
CR2	If $D_1, D_2 \in \mathbf{S}(C)$, $D_1 \sqcap D_2 \sqsubseteq D \in \mathcal{O}$, and $D \notin \mathbf{S}(C)$, then add D to $\mathbf{S}(C)$.
CR3	If $E \in \mathbf{S}(C)$, and $E \sqsubseteq \exists R.D \in \mathcal{O}$, and $\langle C,D\rangle \notin \mathbf{R}(R)$ then add $\langle C,D\rangle$ to $\mathbf{R}(R)$.
CR4	If $\langle E,C\rangle \in \mathbf{R}(R)$, $D_1 \in \mathbf{S}(C)$, $\exists R.D_1 \sqsubseteq D_2 \in \mathcal{O}$, and $D_2 \notin \mathbf{S}(E)$ then add D_2 to $\mathbf{S}(E)$.
CR5	If $\langle C,D\rangle \in \mathbf{R}(R)$, $\bot \in \mathbf{S}(D)$, and $\bot \notin \mathbf{S}(C)$ then add $\bot$ to $\mathbf{S}(C)$.
CR10	If $\langle E,C\rangle \in \mathbf{R}(R)$, $R \sqsubseteq S \in \mathcal{O}$, and $\langle E,C\rangle \notin \mathbf{R}(S)$ then add $\langle E,C\rangle$ to $\mathbf{R}(S)$.
CR11	If $\langle E,C\rangle \in \mathbf{R}(R_1)$, $\langle C,D\rangle \in \mathbf{R}(R_2)$, $R_1 \circ R_2 \sqsubseteq S \in \mathcal{O}$, and $\langle E,D\rangle \notin \mathbf{R}(S)$ then add $\langle E,D\rangle$ to $\mathbf{R}(S)$.

Fig. 4 A subset of the $\mathcal{EL}^{++}$ completion rules [8] specific to $\mathcal{EL}_{\bot}^{+}$ ontologies

the form $\mathbf{S}(C)$ and $\mathbf{R}(R)$ where C is a concept and R is a role. The set $\mathbf{S}(C)$ loosely corresponds to all concepts D such that $C \sqsubseteq D$ is derived by our procedure and the set $\mathbf{R}(R)$ to all pairs $\langle C,D\rangle$ for which $C \overset{R}{\to} D$ is derived by our procedure. The main difference is that the completion-based procedure requires the input ontology $\mathcal{O}$ to be normalized (flattened) so that each concept inclusion contains at most one concept constructor. This can always be done by repeatedly replacing complex concepts with fresh atomic concepts and adding the corresponding equivalences, followed by subsequent simplifications. For example, the ontology in Example 1 can be normalized as follows using fresh atomic concepts X_1 and X_2:

- $A \sqsubseteq \exists R.(C \sqcap D) \quad \rightsquigarrow \quad A \sqsubseteq \exists R.X_1, \quad X_1 \equiv (C \sqcap D) \quad \rightsquigarrow$
  $\rightsquigarrow \quad A \sqsubseteq \exists R.X_1, \quad X_1 \sqsubseteq C, \quad X_1 \sqsubseteq D, \quad C \sqcap D \sqsubseteq X_1;$
- $B \equiv A \sqcap \exists S.D \quad \rightsquigarrow \quad B \equiv A \sqcap X_2, \quad X_2 \equiv \exists S.D \quad \rightsquigarrow$
  $\rightsquigarrow \quad B \sqsubseteq A, \quad B \sqsubseteq X_2, \quad A \sqcap X_2 \sqsubseteq B, \quad X_2 \sqsubseteq \exists S.D, \quad \exists S.D \sqsubseteq X_2;$
- $\exists S.D \sqsubseteq C$ (already normalized);
- $R \sqsubseteq S$ (already normalized).

The subset of $\mathcal{EL}^{++}$ completion rules [8] relevant to $\mathcal{EL}_{\bot}^{+}$ is listed in Fig. 4. Every rule except for CR5 deals with a specific type of a normalized axiom in $\mathcal{O}$.[5] One can easily see similarities with the rules in Fig. 2: the rules correspond respectively to $\mathsf{R}_{\sqsubseteq}$, $\mathsf{R}_{\sqcap}^{+}$, $\mathsf{R}_{\exists}^{-}$, $\mathsf{R}_{\exists}^{+}$, $\mathsf{R}_{\bot}$, R_{H}, and $\mathsf{R}_{\circ}$. Note that there is no rule that is analogous to $\mathsf{R}_{\sqcap}^{-}$ because axioms of the form $C \sqsubseteq D_1 \sqcap D_2$ are replaced with $C \sqsubseteq D_1$ and $C \sqsubseteq D_2$ during normalization.

To compute the entailed subsumptions using completion rules, the algorithm first sets $\mathbf{S}(C) = \{C, \top\}$ and $\mathbf{R}(R) = \emptyset$ for each C and R, and then repeatedly applies the rules until no rule is applicable. Initialization of $\mathbf{S}(C)$ is similar to applying the initialization rules R_0 and $\mathsf{R}_{\top}$ in our case. For our example, e.g., the rules in Fig. 4 produce $\mathbf{S}(A) = \mathbf{S}(B) = \{\top, A, B, X_2, C\}$, $\mathbf{S}(X_1) = \{\top, X_1, C, D\}$, $\mathbf{S}(C) = \{\top, C\}$, $\mathbf{S}(D) = \{\top, D\}$, $\mathbf{S}(X_2) = \{\top, X_2\}$, $\mathbf{R}(R) = \langle A, X_1\rangle$, $\langle B, X_1\rangle$, and $\mathbf{R}(S) = \{\langle A, X_1\rangle, \langle A, D\rangle, \langle B, X_1\rangle, \langle B, D\rangle\}$. It is also possible to apply the completion rules in a goal-directed way by initializing $\mathbf{S}(C)$ dynamically as they become reachable from the goal query [15,51].

It is worth noting that the behavior of the completion-based procedure is sensitive to the chosen normalization method. Typically, equivalences like $X_1 \equiv C \sqcap D$ are introduced for occurrences of $C \sqcap D$ irrespective of whether such occurrences are positive or negative. A polarity-preserving structural transformation (see, e.g., [51]) can be used to introduce weaker axioms depending on polarities of such occurrences. For example, it is sufficient to replace the axiom $A \sqsubseteq \exists R.(C \sqcap D)$ in our example, with just $A \sqsubseteq \exists R.X_1$ and $X_1 \sqsubseteq C \sqcap D$, thus avoiding $C \sqcap D \sqsubseteq X_1$ and subsequent applications of rule CR2. This has a similar effect as restricting our rule $\mathsf{R}_{\sqcap}^{+}$ only to conjunctions that occur negatively. Even if expressions occur

[5] In other formulations [13,15], CR1 and CR2 are generalized to a rule for n-ary conjunctions on the left.

$$\mathsf{Ax}: \frac{}{C \sqsubseteq C} \qquad \mathsf{AndL1}: \frac{C \sqsubseteq E}{C \sqcap D \sqsubseteq E} \qquad \mathsf{AndL2}: \frac{D \sqsubseteq E}{C \sqcap D \sqsubseteq E} \qquad \mathsf{AndR}: \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2}$$

$$\mathsf{Ex}: \frac{C \sqsubseteq D}{\exists R.C \sqsubseteq \exists R.D} \qquad \mathsf{Cut}: \frac{C \sqsubseteq D \quad D \sqsubseteq E}{C \sqsubseteq E} \qquad \mathsf{Concept}: \frac{C \sqsubseteq D \quad E \sqsubseteq F}{C \sqsubseteq F} : D \sqsubseteq E \in \mathcal{O}$$

Fig. 5 Sequent-style rules for $\mathcal{EL}$ [41]

both positively and negatively, it is still possible to avoid introducing full equivalences. For example, our second axiom $B \equiv A \sqcap \exists S.D$, can be normalized to $B \sqsubseteq A$, $B \sqsubseteq \exists S.D$ (for the inclusion $B \sqsubseteq A \sqcap \exists S.D$), and $\exists S.D \sqsubseteq X_2$, $A \sqcap X_2 \sqsubseteq B$ (for the inclusion $A \sqcap \exists S.D \sqsubseteq B$), thus avoiding $X_2 \sqsubseteq \exists S.D$ and subsequent applications of CR3. Reducing the number of normalized axioms, in general, should reduce the number of inferences. However, aggressive normalization makes it difficult to apply some of the optimizations because the connection between the original and the normalized ontology is destroyed. We will discuss this issue in more detail in Section 5 when we talk about some specific optimizations used in ELK.

Our calculus is certainly not the only one that avoids normalization for the input ontology. A similar *proof-theoretic* procedure for $\mathcal{EL}$ (without $\top$) was described by Hofmann [41]. The procedure is based on sequent-style rules in Fig. 5. These rules are similar to our basic rules in Fig. 1. The main difference is that the sequent-style rules introduce new concepts both on the left and on the right of concept subsumptions. This makes it possible to prove the cut elimination result (that Cut rule is not needed), and as a consequence, the subformula property (that all concepts in the rules can be restricted to subconcepts of the goal and $\mathcal{O}$), from which the polynomial complexity of the procedure follows. This is similar to our results in Section 3.4. Note, however, that our rules in Fig. 1 have three cut-like rules, namely $\mathsf{R}_\bot$, $\mathsf{R}_\exists$, $\mathsf{R}_\circ$—just like Cut, these are the only rules that have some concept that does not occur in the conclusion of the rule or in the ontology, but occurs in the premises. These rules, obviously, cannot be removed from the system. Thus our rules have the subformula property (when restricted to $\mathcal{EL}$), but do not enjoy the similar cut-elimination property.

Apart from (re-)establishing the polynomial complexity of $\mathcal{EL}$, the sequent-style procedure of Hofmann does not seem to have any purpose. Once the subformula property is established, there does not appear to be much difference between the rules Cut and Concept. In fact, these two rules make the sequent-style procedures rather inefficient for some quite common situations. For example, if the ontology contains a long chain of concept inclusions $A_{i-1} \sqsubseteq A_i$ $(1 \leq i \leq n)$, the procedure (both with and without the Cut rule) derives all $A_i \sqsubseteq A_j$ with $0 \leq i \leq j \leq n$, and hence there are $O(n^3)$ possible applications of both rules Cut and Concept. Thus, every subsumption is produced, on average, $O(n)$ times by these rules from different premises. This does not happen with our procedure in Fig. 1 since there are only $O(n^2)$ possible applications of $\mathsf{R}_\sqsubseteq$ (in fact every subsumption will be produced exactly once for this example). Of course, in theory, a similar situation can occur with our rule $\mathsf{R}_\exists$, which also has a cubic worst-case complexity in the number of different concepts, but due to the presence of existential restrictions, the rule is significantly more restricted (note that not only E, C, and D are required to occur in the input due to the subformula property, but also $\exists R.C$ and $\exists R.D$) and an example that exhibits the worst case complexity would probably be too artificial to really occur in practice. Our further optimizations, in particular existential links and the notion of redundancy, which effectively prevent the conclusions of $\mathsf{R}_\exists$ to be used as the left premise of $\mathsf{R}_\exists$, make this worst-case situation even less likely.

Algorithm 1: The abstract saturation procedure

```
input   : Input: a set of expressions, R: a set of inference rules
output  : Closure: the closure of Input under R
1  Closure, Todo ← ∅;
2  for expression ∈ Input do                              /* initialize */
3      Todo.add(expression);
4  while Todo ≠ ∅ do                                       /* close */
5      expression ← Todo.takeNextElement();
6      process (expression);
7  return Closure;
8  process(expression) :
9      if expression ∉ Closure then
10         Closure.add(expression);
11         for inference ∈ R(expression, Closure) do
12             Todo.add(inference.conclusion);
```

4 Computing the Closure under the Inference Rules

So far we have discussed theoretical properties of the calculus based on the inference rules in Fig. 3, such as soundness and completeness of the procedure for various reasoning tasks. In this section, we focus on the algorithmic aspect of the problem, specifically, on how to compute the deductive closure under the inference rules in an efficient way. Although it is relatively easy to implement the procedure so that it runs in polynomial time, we will demonstrate that a number of optimizations and efficient data structures can account for a significant speedup in practice, even though they do not reduce the worst-case complexity.

We first give a high-level overview of the procedure, which can essentially be applied to any inference system, and then provide a more detailed description that is specific to the inference rules in Fig. 3. For simplicity, in this section, we do not discuss how to avoid redundant applications of rule $\mathsf{R}^-_\exists$; this will be done later in Section 5.1.

4.1 The Abstract Saturation Procedure

In this section we present a basic algorithm for computing the deductive closure of input expressions under inference rules, which we call the *abstract saturation procedure*. This is a well-known procedure and it is similar to the 'given clause' algorithm (set of support strategy) used in saturation-based theorem proving (see, e.g., [18, 109]) and semi-naive (bottom-up) evaluation of logic programs (see, e.g., [1]).

The abstract saturation procedure can be described using Algorithm 1. In order to compute the deductive closure of a set of expressions Input under a set of inference rules R, the procedure maintains two collections of expressions: the queue Todo contains expressions to which the rules have yet to be applied and the set Closure accumulates the expressions to which the rules are applied. A queue can have duplicate elements whereas a set cannot. Todo is first initialized with the input expressions (lines 2–3). After that, the expressions are repeatedly removed from Todo (in no specific order) and processed by method process(expression) until the queue becomes empty (lines 4–6). Expressions are processed by applying all rules R (expression, Closure) between this expression and the expressions in Closure (lines 11–12). Note that the new conclusions are inserted into Todo, which are again processed in the main loop (lines 4–6). In order to avoid a potentially infinite loop, the rules are applied only the first time an expression is inserted into Closure (see lines 9, 10). This guarantees termination of the algorithm in case there are only finitely many expressions.

Example 5 The derivation in Example 4 already presents the expressions in the order in which they are processed by the abstract saturation algorithm. For example, after processing expression (28), Closure contains expressions (23)–(28), and Todo contains expressions (29) and (30). The algorithm then takes and removes expression (29) from Todo, adds it to Closure, and applies all inferences involving this expression and the previously processed expressions (23)–(28). In this case, no inference rules are applicable. The algorithm then takes the next expression (30) from Todo, adds it to Closure, and applies all inferences involving this expression and the previously processed expressions (23)–(29). In this case, rule $\mathsf{R}_{\exists}^{+}$ is applicable to the premises (26) and (30), so the conclusion (31) is added to Todo.

Correctness of Algorithm 1 is a consequence of the following (semi-)invariants that can be proved by induction over the execution of the algorithm:

(i) Every expression in Todo and Closure is either from Input or obtained by an inference rule from some expressions in Closure;
(ii) Closure does not contain duplicate expressions;
(iii) After every iteration of the loop at lines 4–6:
  (a) Either the size of Closure increases or, otherwise, it remains the same and the size of Todo decreases;
  (b) Every expression in Input occurs in either Todo or Closure;
  (c) Every conclusion of an inference from Closure occurs in either Todo or Closure.

From (i), it follows that Algorithm 1 can add to Closure only expressions that are derivable by the rules from Input. Therefore, from (ii) and (iii).(a), it follows that if there are only finitely many different expressions that can be derived from Input, then Algorithm 1 terminates. Finally, from (iii).(b) and (iii).(c) it follows that when Algorithm 1 terminates (and thus Todo is empty), Closure contains all expressions derivable from Input.

4.2 Implementing the Abstract Saturation Procedure for $\mathcal{EL}_{\bot}^{+}$

A key detail of Algorithm 1 not discussed so far is the application of inference rules in line 11, where the abstract saturation procedure needs to compute all inferences between the given expression and the expressions in Closure. How can we implement this computation? One possibility is to enumerate all tuples of expressions in Closure that contain the given expression,[6] and check applicability of all rules to each tuple. While this can be implemented in polynomial time in the size of Closure (all rules have a bounded number of premises, so it suffices to consider polynomially many tuples of bounded size), this would clearly be impractical. For a more efficient implementation, we need a goal-directed way to determine which rules can be applied to the given premise and with which other premises from Closure.

In this section, we present algorithms and datastructures that enable efficient search and application of the rules from Fig. 3. The optimized saturation procedure for $\mathcal{EL}_{\bot}^{+}$ consists of three phases. The first two phases ‘compile’ axioms to produce an efficient representation of inference rules, and the third phase uses this representation to compute the closure.

[6] Note that the given expression will already be present in Closure at this point.

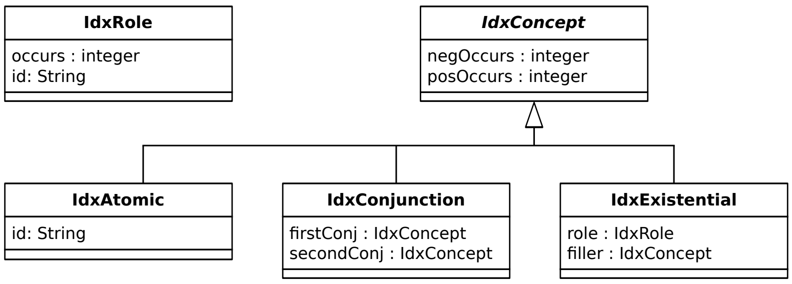

Fig. 6 Classes for indexing objects

4.2.1 Phase 1: Indexing

The goal of the indexing phase is to build a representation of the input ontology $\mathcal{O}$ so that the side conditions of the rules in Fig. 3 can be verified efficiently. For this purpose, we store information about occurrences of roles and concepts in $\mathcal{O}$ using a system of *indexed objects*.

The type hierarchy of indexed objects is shown in Fig. 6; it closely follows the recursive definition of $\mathcal{EL}$ expressions. The top level classes IdxRole and IdxConcept represent roles and concepts, respectively. For each indexed role, the field occurs stores the number of times this role occurs in the ontology $\mathcal{O}$, and the field id contains the string identifier of the role. For indexed concepts we separately keep track of the number of their negative and positive occurrences in $\mathcal{O}$ in the fields negOccurs and posOccurs. These counters are used to determine which expressions occur in the ontology under which polarity, and to update this information when adding or removing axioms. The different types of $\mathcal{EL}$ concepts are represented by the corresponding subclasses of IdxConcept. For example, a conjunction $C \sqcap D$ is represented by an object of type IdxConjunction with fields firstConj and secondConj pointing to the indexed objects that represent C and D, respectively. In addition, there are distinguished instances top and bottom of IdxConcept that represent $\top$ and $\bot$, respectively. From now on, when we mention roles or concepts, we refer to their indexed representations.

Apart from creating the indexed objects, the indexing phase also constructs data structures for efficient look-up of side conditions of the inference rules. Specifically, the following tables (sets of tuples) of indexed objects are constructed.

$$\begin{aligned}
\mathsf{negConjs} &= \{\langle C, D, C \sqcap D\rangle \mid C \sqcap D \text{ occurs negatively in } \mathcal{O}\} \\
\mathsf{negExists} &= \{\langle R, C, \exists R.C\rangle \mid \exists R.C \text{ occurs negatively in } \mathcal{O}\} \\
\mathsf{conclncs} &= \{\langle C, D\rangle \mid C \sqsubseteq D \in \mathcal{O}\} \\
\mathsf{rolelncs} &= \{\langle R, S\rangle \mid R \sqsubseteq S \in \mathcal{O}\} \\
\mathsf{roleComps} &= \{\langle R_1, R_2, S\rangle \mid R_1 \circ R_2 \sqsubseteq S \in \mathcal{O}\}
\end{aligned}$$

Example 6 Consider the ontology $\mathcal{O}$ from Example 1. Its index contains the following indexed objects with occurrence counts as indicated.[7]

IdxRole	R	S
occurs	2	4

IdxConcept	$\top$	$\bot$	A	B	C	D	$C \sqcap D$	$\exists R.(C \sqcap D)$	$\exists S.D$	$A \sqcap \exists S.D$
negOccurs	0	0	2	1	0	2	0	0	2	1
posOccurs	0	0	1	1	2	2	1	1	1	1

[7] Recall that we treat the equivalence axiom (ax2) as two individual concept inclusion axioms, so, e.g., the occurrence count of S is 4, not 3.

The look-up tables contain the following tuples.

$$\begin{aligned}
\mathsf{negConjs} &= \{\langle A, \exists S.D, A \sqcap \exists S.D\rangle\} \\
\mathsf{negExists} &= \{\langle S, D, \exists S.D\rangle\} \\
\mathsf{conclncs} &= \{\langle A, \exists R.(C \sqcap D)\rangle, \langle B, A \sqcap \exists S.D\rangle, \langle A \sqcap \exists S.D, B\rangle, \langle \exists S.D, C\rangle\} \\
\mathsf{roleIncs} &= \{\langle R, S\rangle\} \\
\mathsf{roleComps} &= \emptyset
\end{aligned}$$

To give an idea how these index datastructures can be used during the saturation phase, consider the point in the derivation from Example 4 when the saturation algorithm processes the expression $A \sqsubseteq \exists S.D$ in line (31). To apply rule $\mathsf{R}^+_\sqcap$ to this expression, the algorithm iterates over those tuples in negConjs whose first or second component is $\exists S.D$ to find all possible ways to satisfy the side condition. Since negConjs contains $\langle A, \exists S.D, A \sqcap \exists S.D\rangle$, the algorithm checks if Closure already contains $A \sqsubseteq A$, which can be used as the first premise of $\mathsf{R}^+_\sqcap$. Since this is the case, the conclusion $A \sqsubseteq A \sqcap \exists S.D$ is added to the Todo queue. Note that (a pointer to) the indexed conjunction $A \sqcap \exists S.D$ used in the conclusion can be taken directly from the table negConjs. This illustrates that conclusions of inference rules can be constructed by simply following the pointers in the index, and no new indexed objects need to be created during the saturation phase.

Indexing is a lightweight task that can be performed by a single recursive traversal through the structure of each axiom in the ontology. Furthermore, the index datastructures can be constructed incrementally by considering one axiom at a time. Since we keep the exact counts of negative and positive occurrences of concepts and roles, the index can be easily updated not only when axioms are added, but also when axioms are removed. This can be highly useful when working with ontologies that are being modified, e.g., in ontology editors, as it is sufficient to reindex changed axioms only.

Example 7 Let us see how the index data structures in Example 6 are updated if we remove the axiom $B \equiv A \sqcap \exists S.D$ from $\mathcal{O}$. First, the tuples $\langle B, A \sqcap \exists S.D\rangle$ and $\langle A \sqcap \exists S.D, B\rangle$ are removed from the table conclncs. Second, the counters for both negative and positive occurrences of B, $A \sqcap \exists S.D$, A, $\exists S.D$, and D are decremented, and the occurrence counter for S is decreased by 2. Since the counter for negative occurrences of the conjunction $A \sqcap \exists S.D$ becomes zero, the tuple $\langle A, \exists S.D, A \sqcap \exists S.D\rangle$ is removed from the table negConjs. The negative occurrence count of the existential restriction $\exists S.D$ remains positive (the concept still occurs negatively in (ax3)), so the table negExists remains unchanged. Finally, since there are no more occurrences of B and $A \sqcap \exists S.D$, the corresponding indexed objects are deleted.

4.2.2 Phase 2: Saturation of Roles

In order to efficiently check the side conditions of rules $\mathsf{R}^+_\exists$ and $\mathsf{R}_\circ$, we compute the reflexive transitive closure $\sqsubseteq^*_\mathcal{O}$ of the role inclusion axioms of $\mathcal{O}$, and store it in a table called hier (for role hierarchy):

$$\mathsf{hier} = \{\langle R, S\rangle \mid R \sqsubseteq^*_\mathcal{O} S\}.$$

The table hier can be easily computed from the table roleIncs obtained in the previous phase. Since the number of roles in real-world ontologies is usually much smaller than the number of concepts, any reasonable algorithm for computing the transitive closure can be used with no significant impact on overall performance.

R_0:	If	$\mathsf{inits}(C)$,
	then	$\mathsf{subs}(C,C)$.
$\mathsf{R}_\top$:	If	$\mathsf{inits}(C)$ & $\mathsf{top.negOccurs} > 0$,
	then	$\mathsf{subs}(C,\mathsf{top})$.
$\mathsf{R}_\bot$:	If	$\mathsf{links}(E,R,C)$ & $\mathsf{subs}(C,\mathsf{bottom})$,
	then	$\mathsf{subs}(E,\mathsf{bottom})$.
$\mathsf{R}^-_\sqcap$:	If	$\mathsf{subs}(C,D)$ & D **instanceOf** IdxConjunction,
	then	$\mathsf{subs}(C,D.\mathsf{firstConj})$ and $\mathsf{subs}(C,D.\mathsf{secondConj})$.
$\mathsf{R}^+_\sqcap$:	If	$\mathsf{subs}(C,D_1)$ & $\mathsf{subs}(C,D_2)$ & $\mathsf{negConjs}(D_1,D_2,D)$,
	then	$\mathsf{subs}(C,D)$.
$\mathsf{R}^-_\exists$:	If	$\mathsf{subs}(C,D)$ & D **instanceOf** IdxExistential,
	then	$\mathsf{links}(C,D.\mathsf{role},D.\mathsf{filler})$.
$\mathsf{R}^+_\exists$:	If	$\mathsf{links}(E,R,C)$ & $\mathsf{subs}(C,D)$ & $\mathsf{negExists}(S,D,F)$ & $\mathsf{hier}(R,S)$,
	then	$\mathsf{subs}(E,F)$.
$\mathsf{R}_\sqsubseteq$:	If	$\mathsf{subs}(C,D)$ & $\mathsf{concIncs}(D,E)$,
	then	$\mathsf{subs}(C,E)$.
$\mathsf{R}_\circ$:	If	$\mathsf{links}(E,R_1,C)$ & $\mathsf{links}(C,R_2,D)$ & $\mathsf{roleComps}(S_1,S_2,S)$ & $\mathsf{hier}(R_1,S_1)$ & $\mathsf{hier}(R_2,S_2)$,
	then	$\mathsf{links}(E,S,D)$.
$\mathsf{R}_\rightsquigarrow$:	If	$\mathsf{links}(E,R,C)$,
	then	$\mathsf{inits}(C)$.

Fig. 7 The closure properties induced by rules in Fig. 3

Example 8 The reflexive transitive closure of the table roleIncs in Example 6 is

$$\mathsf{hier} = \{\langle R,R\rangle, \langle R,S\rangle, \langle S,S\rangle\}.$$

4.2.3 Phase 3: Saturation of Concepts

In this phase, we compute the saturation of Input under the inference rules in Fig. 3 using a specialized version of Algorithm 1. Recall that our inference rules operate with three types of expressions: $\mathsf{init}(C)$, $C \sqsubseteq D$, and $E \overset{R}{\to} C$. Depending on where these expressions are saved, we use different representations for them. The expressions in Todo are represented by objects with the corresponding number of parameters, whereas the expressions in Closure are represented using three tables inits, subs, and links for the respective types of expressions as given in Table 2.

Before presenting details of the saturation algorithm, we first reformulate in Fig. 7 the rules from Fig. 3 as closure properties in the above representation. In these rule formulations, for each table T and tuple x the expression $T(x)$ stands for $x \in T$.

The main body of the saturation algorithm remains the same as for the abstract saturation procedure (Algorithm 1, lines 2–6). We only replace the implementation of the function `process`(expression) as shown in Algorithm 2. Each expression is processed according to its type, corresponding to all possible ways the expression can be used as a premise of the rules in Fig. 7.

4.3 Efficient Join Computation

Iterations over joins of tables are used extensively in Algorithm 2 to retrieve all matching side conditions and premises of inference rules. We can optimize such iterations by iter-

Algorithm 2: process(expression)

```
process(Init($C$)) :
    if $C \notin$ inits then
        inits.add($C$);
        Todo.add(new Sub($C,C$));                                   // rule $\mathsf{R}_0$
        if top.negOccurs $> 0$ then
            Todo.add(new Sub($C$, top));                            // rule $\mathsf{R}_\top$

process(Sub($C,D$)) :
    if $\langle C,D\rangle \notin$ subs then
        subs.add($\langle C,D\rangle$);
        if $D =$ bottom then
            for each $E,R$ with links($E,R,C$) do
                Todo.add(new Sub($E$, bottom));                     // rule $\mathsf{R}_\bot$
        if $D$ instanceOf IdxConjunction then
            Todo.add(new Sub($C,D$.firstConj));
            Todo.add(new Sub($C,D$.secondConj));                    // rule $\mathsf{R}_\sqcap^-$
        for each $D_2,E$ with subs($C,D_2$) and negConjs($D,D_2,E$) do
            Todo.add(new Sub($C,E$));                               // rule $\mathsf{R}_\sqcap^+$
        for each $D_1,E$ with subs($C,D_1$) and negConjs($D_1,D,E$) do
            Todo.add(new Sub($C,E$));                               // rule $\mathsf{R}_\sqcap^+$
        if $D$ instanceOf IdxExistential then
            Todo.add(new Link($C,D$.role,$D$.filler));              // rule $\mathsf{R}_\exists^-$
        for each $E,F,R,S$ with links($E,R,C$) and negExists($S,D,F$) and hier($R,S$) do
            Todo.add(new Sub($E,F$));                               // rule $\mathsf{R}_\exists^+$
        for each $E$ with conclncs($D,E$) do
            Todo.add(new Sub($C,E$));                               // rule $\mathsf{R}_\sqsubseteq$

process(Link($E,R,C$)) :
    if $\langle E,R,C\rangle \notin$ links then
        links.add($\langle E,R,C\rangle$);
        if subs($C$, bottom) then
            Todo.add(new Sub($E$, bottom));                         // rule $\mathsf{R}_\bot$
        for each $D,F,S$ with subs($C,D$) and negExists($S,D,F$) and hier($R,S$) do
            Todo.add(new Sub($E,F$));                               // rule $\mathsf{R}_\exists^+$
        for each $D,R_2,S_1,S_2,S$ with links($C,R_2,D$) and roleComps($S_1,S_2,S$) and hier($R,S_1$) and
        hier($R_2,S_2$) do
            Todo.add(new Link($E,S,D$));                            // rule $\mathsf{R}_\circ$
        for each $D,R_1,S_1,S_2,S$ with links($D,R_1,E$) and roleComps($S_1,S_2,S$) and hier($R_1,S_1$)
        and hier($R,S_2$) do
            Todo.add(new Link($D,S,C$));                            // rule $\mathsf{R}_\circ$
        Todo.add(new Init($C$));                                    // rule $\mathsf{R}_\rightsquigarrow$
```

Table 2 Representation of $\mathcal{EL}_\bot^+$ expressions

Expression	Representation in Todo	Representation in Closure
init(C)	Init(C)	$C \in$ inits
$C \sqsubseteq D$	Sub(C,D)	$\langle C,D\rangle \in$ subs
$E \overset{R}{\to} C$	Link(E,R,C)	$\langle E,R,C\rangle \in$ links

ating over smaller tables and precomputing partial joins. In this section we illustrate these techniques on the application of rules $\mathsf{R}^+_\sqcap$ and $\mathsf{R}^+_\exists$.

Consider rule $\mathsf{R}^+_\sqcap$ implemented in Algorithm 2 in lines 16–19. To apply the rule with an axiom $C \sqsubseteq D$ as the first premise, the loop in line 16 iterates over all indexed concepts D_2 and E such that $\mathsf{subs}(C, D_2) \wedge \mathsf{negConjs}(D, D_2, E)$ holds, i.e., such that the subsumption $C \sqsubseteq D_2$ has already been processed and $E = D \sqcap D_2$ occurs negatively in the ontology. One possibility is to iterate over all D_2 such that $\mathsf{subs}(C, D_2)$ holds, and for each of them check if a (necessarily unique) E with $\mathsf{negConjs}(D, D_2, E)$ exists. Another possibility is to iterate over all D_2 such that $\mathsf{negConjs}(D, D_2, E)$ holds (for $E = D \sqcap D_2$) and for each of them check if $\mathsf{subs}(C, D_2)$ holds. Depending on which iteration is shorter, one method is more efficient than the other. For this reason, we do not fix the order of iteration upfront; instead, we dynamically choose the order that results in iterating over the smaller set.

The above approach can be further improved for rules where the same join is computed multiple times. Consider the application of rule $\mathsf{R}^+_\exists$ to a link $E \overset{R}{\to} C$ as the first premise implemented in Algorithm 2 in lines 31–32. For simplicity, we ignore the role hierarchy in this discussion. In that case, the loop in line 31 iterates over all D such that $\mathsf{subs}(C, D) \wedge \mathsf{negExists}(R, D, F)$ holds (for $F = \exists R.D$), i.e., such that the subsumption $C \sqsubseteq D$ has already been processed and $F = \exists R.D$ occurs negatively in the ontology. Since this iteration is independent of E, the algorithm repeats it for very link $E \overset{R}{\to} C$ with the same R and C. To avoid the repetition, we cache the result as 'propagations' $\mathsf{props}(R, C, F) := \mathsf{subs}(C, D) \bowtie \mathsf{negExists}(R, D, F)$, which is updated every time a new subsumption $C \sqsubseteq D$ is derived.

4.4 Relations with Implementations of Completion-Based Procedures

As we mentioned in the beginning of Section 4.1, algorithms for computing the closure under inference rules are well studied and commonly used in many areas of computer science. So it is not surprising that other rule-based reasoners implement procedures similar to Algorithm 1. In this section we point out the main differences between our implementation and the implementation of completion-based procedures first used in the reasoner CEL [14], and later adapted by other reasoners such as jcel [72], Snorocket [66], and Cheetah [94].

Recall from Section 3.6, that completion-based procedures work with ontologies containing only normalized axioms. The procedure computes sets $\mathbf{S}(C)$ and $\mathbf{R}(R)$ for every atomic concept C and role R by applying the rules in Fig. 4 starting from the initial values $\mathbf{S}(C) = \{\top, C\}$ and $\mathbf{R}(R) = \emptyset$. Like in our case, the CEL procedure uses queues for controlled application of inference rules. Unlike our procedure, however, the queues collect not conclusions of applied rules, but expressions of the form $A_1 \sqcap \cdots \sqcap A_n \to D$ that represent 'remainders' of the normalized axioms containing the produced subsumer on the left-hand side [13,15]. For example, when a new subsumer D_1 is added to $\mathbf{S}(C)$, and the ontology contains an axiom $D_1 \sqcap D_2 \sqcap D_3 \sqsubseteq D$, the procedure adds the remainder '$D_2 \sqcap D_3 \to D$' for D_1 in this axiom to the queue of C. When this remainder is processed, it is checked whether $\mathbf{S}(C)$ contains both D_2 and D_3, and if so, D is added to $\mathbf{S}(C)$, which can result in remainders for D being added to the queue for C in a similar way. This strategy was inspired by the linear-time algorithm for checking satisfiability of propositional Horn formulas [30].

The CEL procedure can be alternatively described using the inference rules in Fig. 8. Intuitively, the rules deal with negative occurrences of conjunctions by deriving intermediate subsumers of the form $D_i \to D$ (we consider only binary conjunctions, so there is at most one concept on the left of $\to$). The procedure thus can be seen as an implementation of Algorithm 1 for these rules, except that the intermediate subsumers produced by the first

$$\frac{C \sqsubseteq D_1}{C \sqsubseteq (D_2 \to D)} : D_1 \sqcap D_2 \sqsubseteq D \in \mathcal{O} \qquad \frac{C \sqsubseteq D_2}{C \sqsubseteq (D_1 \to D)} : D_1 \sqcap D_2 \sqsubseteq D \in \mathcal{O} \qquad \frac{C \sqsubseteq (D_2 \to D) \quad C \sqsubseteq D_2}{C \sqsubseteq D}$$

Fig. 8 Processing negative conjunctions in CEL using intermediate implications

two rules are not saved in Closure, and the conclusion $C \sqsubseteq D$ produced by the last rule are not inserted into Todo, but processed immediately. In order to optimize applications of first two rules, CEL precomputes remainders for each concept D occurring on the left-hand side of axioms, and saves them in a set $\hat{O}(D)$. This is similar to indexing of axioms in our case.

The first two rules in Fig. 8, however, may result in many unnecessary inferences. Some concepts can occur in ontologies in many conjunctions. For example, in SNOMED CT there are several concepts that occur in over 1,000 negative conjunctions. Typically, these are general concepts, such as 'Drug' or 'Disease', that are used to define more specific concepts, e.g., HeartDisease $\equiv$ Disease $\sqcap \exists$affects.Heart, LiverDisease $\equiv$ Disease $\sqcap \exists$affects.Liver. Since such concepts are general (high in the class hierarchy), they can be derived as subsumers of many concepts, e.g., HeartDisease $\sqsubseteq$ Disease, LiverDisease $\sqsubseteq$ Disease, so the first two rules in Fig. 8 can be applied very often. Each rule application produces as many intermediate conclusions as there are conjunctions containing the subsumer. Thus, if the subsumer occurs in n conjunctions, and was derived as a subsumer of m concepts, there will be $n \cdot m$ conclusions. This can be a big number for large ontologies, such as SNOMED CT.

Cheetah [94] has experimented with a slightly different approach of applying completion rules. Instead of deriving intermediate 'remainders' of axioms, the reasoner counts down how many subsumers on the left-hand side of axioms are remained to be satisfied. That is, for every concept C and every axiom $\alpha = A_1 \sqcap \cdots \sqcap A_n \sqsubseteq D \in \mathcal{O}$ the procedure stores a counter $\#(C, \alpha)$ representing the number of concepts A_i such that the subsumption $C \sqsubseteq A_i$ was not yet derived. When a new subsumption $C \sqsubseteq A_i$ is derived, this counter is decremented, and when it reaches 0, the subsumption $C \sqsubseteq D$ is produced. This approach does not require producing the actual 'remainders' of axioms, but requires storing and updating the counters for every C and α. Cheetah has also experimented with a simplified strategy, in which no counters are used, but instead, every time $C \sqsubseteq A$ is derived, for every axiom $\alpha = A_1 \sqcap \cdots \sqcap A_n \sqsubseteq D \in \mathcal{O}$ having $A = A_i$ for some i $(1 \leq i \leq n)$, it is checked whether $C \sqsubseteq A_i$ was already derived for every i $(1 \leq i \leq n)$, in which case $C \sqsubseteq D$ is produced. This strategy has, in fact, outperformed the counter-based strategy on existing ontologies, possibly due to a relatively high overhead of storing and updating the counters and relatively short conjunctions.

The second strategy of Cheetah is similar to our first method of join evaluation for rule $\mathsf{R}^+_\sqcap$ discussed in Section 4.3, where we first iterate over all negative conjunctions containing the derived subsumer, and then check if subsumption with the other conjunct is already derived. As we mentioned, this iteration is not very efficient if the subsumer occurs in many negative conjunctions. Therefore, similar to CEL, both strategies of Cheetah will have problems in this case. In our implementation, however, it is likely that the second method of join evaluation will be selected in this case. That is, we first iterate over all existing subsumers and then check if the conjunction with the produced subsumer occurs negatively in the ontology. This method should work faster if the number of subsumers is small compared to the number of matching negative conjunctions.

Another subtle difference between our procedure and the CEL procedure is processing of 'edges' $\langle C, D \rangle \in \mathbf{R}(R)$, which are created whenever an existential $\exists R.D$ is derived for C. Unlike links in our case, CEL does not insert the edges into the queue, but processes them immediately in a recursive call. This, however, may result in an unbounded recursion (and subsequent stack overflow), especially when implementing the rule for role compositions. In Algorithm 1 we do not have recursive methods, so this problem does not occur. Cheetah also avoids this problem by creating intermediate existential restrictions for the produced edges, which are then inserted into the queue as ordinary subsumers. This would be similar had we implemented our procedure using the simplified rules in Fig. 1.

5 Redundancy and Other Optimizations

The basic approach presented in Section 4 may already work well in practice, but the procedure can be optimized even further. In this section we describe several such optimizations implemented in ELK that turned out to work well in practice. In Section 5.1 we discuss how to avoid some redundant applications of the decomposition rules $\mathsf{R}^-_\exists$ and $\mathsf{R}^-_\sqcap$. In Section 5.2 we describe an optimization for the role composition rule $\mathsf{R}_\circ$. In Section 5.3, we present an optimized treatment of disjointness axioms, which does not require (possibly quadratic) binarization. Finally, in Section 5.4 we describe an optimized procedure for taxonomy construction. We empirically evaluate the effect of all these optimizations in Section 8.

5.1 Optimization of Decomposition Rules

The saturation procedure from the previous section computes the full closure under the inference rules in Fig. 3 without taking advantage of the notion of redundancy from Definition 5. Although it is certainly possible to add the redundancy conditions as additional negative premises for rule $\mathsf{R}^-_\exists$, it is not clear how to check them efficiently. Instead, in ELK we use the following weaker optimization $O_\exists$, which is easy to implement.

$O_\exists$ Do not apply rule $\mathsf{R}^-_\exists$ to a subsumption $E \sqsubseteq \exists S.D$ that has been derived by rule $\mathsf{R}^+_\exists$.

If a subsumption $E \sqsubseteq \exists S.D$ has been derived by rule $\mathsf{R}^+_\exists$, then some premises $E \xrightarrow{R} C$ and $C \sqsubseteq E$ with $R \sqsubseteq^*_\mathcal{O} S$ of the rule must have been derived earlier, so the application of rule $\mathsf{R}^-_\exists$ to $E \sqsubseteq \exists S.D$ is redundant according to Definition 5.

Example 9 Using optimization $O_\exists$, the application of rule $\mathsf{R}^-_\exists$ to axiom (31) in Example 4 is redundant. This avoids deriving expressions (33), (36), and (37), and, in particular, even avoids the need to initialize D.

Interestingly, we can formulate an analogous optimization for rule $\mathsf{R}^-_\sqcap$:

$O_\sqcap$ Do not apply rule $\mathsf{R}^-_\sqcap$ to a subsumption $C \sqsubseteq D_1 \sqcap D_2$ that has been derived by rule $\mathsf{R}^+_\sqcap$.

If $C \sqsubseteq D_1 \sqcap D_2$ has been derived by rule $\mathsf{R}^+_\sqcap$, then both the premises $C \sqsubseteq D_1$ and $C \sqsubseteq D_2$ of the rule must have been derived earlier, so it is clearly redundant to derive them again in rule $\mathsf{R}^-_\sqcap$. Thus, unlike $O_\exists$, optimization $O_\sqcap$ does not affect the set of expression derived during saturation, it can only reduce the number of rule applications.

To implement $O_\sqcap$ and $O_\exists$ in the saturation algorithm, we introduce a new expression $\mathsf{Sub}^+(C,D)$ and change the implementation of rules $\mathsf{R}^+_\sqcap$ and $\mathsf{R}^+_\exists$ in Algorithm 2 to produce Sub^+ instead of Sub. Processing of $\mathsf{Sub}^+(C,D)$ is analogous to $\texttt{process}(\mathsf{Sub}(C,D))$ in Algorithm 2 except that the applications of rules $\mathsf{R}^-_\sqcap$ and $\mathsf{R}^-_\exists$ are omitted.

5.2 Optimization of the Role Composition Rule

All the inference rules in Fig. 3 have the property that if a conclusion is derivable from a premise $E \xrightarrow{S} D$, then the same conclusion is also derivable from each stronger premise $E \xrightarrow{S'} D$ with $S' \sqsubseteq^*_{\mathcal{O}} S$. Therefore, the former is superfluous in the presence of the latter. In this section, we present an optimization of rule $\mathsf{R}_\circ$ that avoids deriving the conclusion $E \xrightarrow{S} D$ from the premises $E \xrightarrow{R_1} C$ and $C \xrightarrow{R_2} D$ if a stronger conclusion $E \xrightarrow{S'} D$ with $S' \sqsubseteq^*_{\mathcal{O}} S$ is also derivable by $\mathsf{R}_\circ$ from the same premises.

First, to simplify the application of rule $\mathsf{R}_\circ$, we expand all indexed role composition axioms under the role hierarchy, i.e., we precompute the join

$$\mathsf{hierComps}(R_1,R_2,S) := \mathsf{roleComps}(S_1,S_2,S) \bowtie \mathsf{hier}(R_1,S_1) \bowtie \mathsf{hier}(R_2,S_2),$$

which can be done already during the role saturation phase. The application of rule $\mathsf{R}_\circ$ in Fig. 7 then simplifies to the following:

$\mathsf{R}_\circ$: If $\mathsf{links}(E,R_1,C)$ & $\mathsf{links}(C,R_2,D)$ & $\mathsf{hierComps}(R_1,R_2,S)$,
then $\mathsf{links}(E,S,D)$.

Second, we eliminate all entries $\mathsf{hierComps}(R_1,R_2,S)$ for which there exists another entry $\mathsf{hierComps}(R_1,R_2,S')$ with $S' \sqsubseteq^*_{\mathcal{O}} S$; to avoid the problem of simultaneously removing all entries with equivalent roles, the entries need to be removed sequentially one at a time. This optimization prevents the algorithm from deriving two different conclusions $E \xrightarrow{S} D$ and $E \xrightarrow{S'} D$ with $S' \sqsubseteq^*_{\mathcal{O}} S$ by rule $\mathsf{R}_\circ$ from the same premises. We denote this optimization $O_\circ$.

It is also possible to implement a more aggressive optimization that discards each derived link $E \xrightarrow{S} D$ in case a stronger link $E \xrightarrow{S'} D$ with $S' \sqsubseteq^*_{\mathcal{O}} S$ has already been processed by the saturation algorithm. Although less restrictive, the practical advantage of $O_\circ$ is that, at a relatively small cost of preprocessing the table $\mathsf{hierComps}$ in the role saturation phase, it avoids even the construction of the weaker link $E \xrightarrow{S} D$ in the concept saturation phase.

5.3 Disjointness Axioms

In OWL EL one can write axioms DisjointClasses($C_1 \ \ldots \ C_n$) meaning that the concepts $C_1,\ldots,C_n$ are *pairwise* disjoint. Although such statements can be straightforwardly translated to DL axioms $C_i \sqcap C_j \sqsubseteq \bot$ for $1 \le i < j \le n$, this translation introduces $n(n-1)/2$ axioms, which can be inefficient for large n.

We can treat large disjointness axioms (whose length exceeds a certain threshold) differently. For each such axiom $\alpha = \text{DisjointClasses}(C_1,\ldots,C_n)$ we introduce a special 'marker' concept D_α and assert $C_i \sqsubseteq D_\alpha$ for $1 \le i \le n$. Then, in the saturation phase, whenever we process a subsumption $C \sqsubseteq D_\alpha$ the second time, i.e., whenever we attempt to insert $\langle C,D_\alpha\rangle$ into table subs in Algorithm 2 but the tuple is already there, we know that $C \sqsubseteq C_i$ must have already been derived for at least two different indexes i, so we derive the conclusion $C \sqsubseteq \bot$.

5.4 Taxonomy Construction

Recall that, for the classification task, we initialize the saturation algorithm with $\mathsf{init}(A)$ for each atomic concept A, and then, by our completeness result, the result of the saturation contains all entailed subsumptions $A \sqsubseteq B$ between atomic concepts. The computed saturation, however, is not very convenient for navigating over subsumptions. Instead, subsumptions are usually represented in the form of a *taxonomy*, which contains equivalent classes of atomic concepts and direct subsumption relations between them. We say that an atomic concept A is *directly subsumed* by an atomic concept B w.r.t. $\mathcal{O}$ if $\mathcal{O} \models A \sqsubseteq B$ and for every atomic concept C such that $\mathcal{O} \models A \sqsubseteq C \sqsubseteq B$, either $\mathcal{O} \models C \equiv A$ or $\mathcal{O} \models C \equiv B$. The procedure of computing the 'direct part' of a transitive relation is usually called *transitive reduction*.

In this section we discuss how to perform transitive reduction of a (transitively closed) set subs of all subsumption relations between atomic concepts. To perform the reduction, for every atomic concept A we compute the set of its equivalent concepts A.equivalent and the set of its direct subsumers A.directSubs.

A naive algorithm for computing A.directSubs is shown in Algorithm 3. The algorithm iterates over all subsumers C of A, and for each of them checks if another subsumer B of A exists with $A \sqsubseteq B \sqsubseteq C$. If no such B exists, then C is a direct subsumer of A. Note that this approach does not work as expected when A has two equivalent subsumers, in which case none of them would be found as direct. Apart from this shortcoming, the algorithm is also inefficient because it performs two nested iterations over the subsumers of A. In realistic ontologies, the number of all subsumers of A can be sizeable, while the number of direct subsumers is usually much smaller, often just one. A more efficient algorithm would take advantage of this and perform the inner iteration only over the set of direct subsumers of A that have been found so far, as shown in Algorithm 4. Note that it is safe to execute Algorithm 4 in parallel for multiple concepts A.

Algorithm 3: Naive Transitive Reduction

```
1  for each $C$ with subs($A$,$C$) do
2      if $C \neq A$ then
3          isDirect ← true;
4          for each $B$ with subs($A$,$B$) do
5              if $B \neq A$ and $B \neq C$ and subs($B$,$C$) then
6                  isDirect ← false;
7          if isDirect then
8              $A$.directSubs.add($C$);
```

Algorithm 4: Optimized Transitive Reduction

```
1  for each $C$ with subs($A$,$C$) do
2      if subs($C$,$A$) then
3          $A$.equivalent.add($C$);
4      else
5          isDirect ← true;
6          for $B \in A$.directSubs do
7              if subs($B$,$C$) then
8                  isDirect ← false;
9                  break;
10             if subs($C$,$B$) then
11                 $A$.directSubs.remove($B$);
12         if isDirect then
13             $A$.directSubs.add($C$);
```

The rest of the taxonomy construction is straightforward. We introduce one taxonomy node for each distinct class of equivalent concepts, and connect the nodes according to the direct subsumption relation. Care has to be taken to put the top and the bottom node in their proper positions, even if $\top$ or $\bot$ do not occur in the ontology.

Example 10 Consider again the ontology from Example 1. The saturation algorithm computes the following subsumptions (projected to atomic concepts):

$$\mathsf{subs} = \{\langle A,A\rangle, \langle A,B\rangle, \langle A,C\rangle, \langle B,A\rangle, \langle B,B\rangle, \langle B,C\rangle, \langle C,C\rangle, \langle D,D\rangle\},$$

from which Algorithm 4 computes the following taxonomy (shown to the right):

$$\begin{aligned}
&A.\mathsf{equivalent} = \{A,B\}, && A.\mathsf{directSubs} = \{C\},\\
&B.\mathsf{equivalent} = \{A,B\}, && B.\mathsf{directSubs} = \{C\},\\
&C.\mathsf{equivalent} = \{C\}, && C.\mathsf{directSubs} = \emptyset,\\
&D.\mathsf{equivalent} = \{D\}, && D.\mathsf{directSubs} = \emptyset.
\end{aligned}$$

$$\begin{tikzcd}
\boxed{C} \arrow[r] & \boxed{\top} & \\
& & \boxed{D} \arrow[ul] \\
\boxed{A \equiv B} \arrow[uu] & \boxed{\bot} \arrow[l] \arrow[ur] &
\end{tikzcd}$$

5.5 Optimizing Completion-Based Procedures

Many of the above optimizations are not specific to our procedure but can be adapted to the completion-based setting as well, and thus potentially benefit other reasoners. For example, our handling of large disjointness axioms from Section 5.3 can be readily applied both in completion- and even tableau-based reasoning.

The optimized implementation $O_\circ$ of the role composition rule $\mathsf{R}_\circ$ from Section 5.2 leads straightforwardly to an analogous optimization of the completion rule CR11 in Fig. 4. Note, however, that this optimization is meaningful only if one takes the view that an edge $\langle C,D\rangle \in \mathbf{R}(R)$ also represents all edges $\langle C,D\rangle \in \mathbf{R}(S)$ with $R \sqsubseteq^*_{\mathcal{O}} S$ and adjusts all completion rules accordingly, similarly as we did in Section 3.5.

The optimizations of redundant decomposition rules $\mathsf{R}^-_\exists$ and $\mathsf{R}^-_\sqcap$ from Section 5.1 are more problematic. It is still possible to formulate a redundancy condition for rule CR3 analogous to that of rule $\mathsf{R}^-_\exists$ in Definition 1 (resp. Definition 5):

> It is redundant to apply rule CR3 to $E \in \mathbf{S}(C)$ and $E \sqsubseteq \exists R.D \in \mathcal{O}$ if $\langle C,F\rangle \in \mathbf{R}(R)$ and $D \in \mathbf{S}(F)$ for some F.

A practical implementation of (a sufficient condition for) redundancy, like $O_\exists$ is, however, more problematic, due to normalization. A straightforward reformulation of $O_\exists$ for completion rules would give the following sufficient condition:

> It is redundant to apply rule CR3 to $E \in \mathbf{S}(C)$ and $E \sqsubseteq \exists R.D \in \mathcal{O}$ if the premise $E \in \mathbf{S}(C)$ was produced by rule CR4 applied to $\langle C,F\rangle \in \mathbf{R}(R)$, $D \in \mathbf{S}(F)$, and $\exists R.D \sqsubseteq E \in \mathcal{O}$ for some F.

To implement this optimization, however, it is necessary to keep track not only of the rule by which $E \in \mathbf{S}(C)$ was derived, but also of the premise $\exists R.D \sqsubseteq E \in \mathcal{O}$ that has been used, since there can be several such axioms with the same E. Further, the optimization works only if existential restrictions were normalized using equivalences $E \equiv \exists R.D$. If more aggressive normalizations are used, such as those mentioned in Section 3.6, the optimization may not work. For example, if axiom $A \equiv B \sqcap \exists R.D$ is normalized like $A \sqsubseteq B$, $A \sqsubseteq \exists R.D$, $\exists R.D \sqsubseteq E$, $B \sqcap E \sqsubseteq A$, and we derive $\langle C,F\rangle \in \mathbf{R}(R)$, $D \in \mathbf{S}(F)$, and $B \in \mathbf{S}(C)$, then we obtain $E \in \mathbf{S}(C)$

by CR4, and $A \in \mathbf{S}(C)$ by CR2, to which CR3 using $A \sqsubseteq \exists R.D \in \mathcal{O}$ is not blocked by the optimization. If instead of $A \sqsubseteq \exists R.D$ we had produced two axioms $A \sqsubseteq E$ and $E \sqsubseteq \exists R.D$ by normalization, then a similar inference for $E \in \mathbf{S}(C)$ would have been blocked.

The problem of efficient taxonomy construction has been well studied in the literature (see, e.g., [11,34]). The CEL reasoner, in particular, employs an optimized procedure similar to Algorithm 4, which also iterates in the inner loop over direct subsumers instead of all subsumers [15]. However, there is a subtle difference between the two procedures. In Algorithm 4 we iterate in the inner loop over the (current) direct subsumers of A (see line 6) and update these direct subsumers using the new subsumer C. The CEL algorithm, on the other hand, iterates over direct subsumers of C and 'marks' all such subsumers. The direct subsumers of A are then determined as those subsumers of A that have not been marked. Practically, this means that the direct subsumers of all strict subsumers C of A should be computed first. CEL does that in a recursive call of the method (which is again dangerous due to a potential stack overflow). Algorithm 4, on the other hand, can compute the direct subsumers of A without need to compute direct subsumers of other concepts. This makes Algorithm 4 better suitable for goal-directed transitive reduction (when direct subsumers need to be computed only for some given concepts, but not all of them), and also for parallel taxonomy construction (when direct subsumers of different concepts can be computed independently of each other).

6 Concurrent Saturation

In Section 5 we have presented several optimization techniques which can make ontology classification faster by reducing the number of operations to perform. However, it is also possible to accelerate the classification procedure by performing several operations in parallel. This way one can effectively make use of multi-core and multi-processor systems that become increasingly widespread. In this section, we discuss the key modifications of our saturation algorithm that are necessary for this. First, in Section 6.1 we discuss some common pitfalls about concurrent algorithms and datastructures, then in Section 6.2 we present a concurrent modification of our general saturation procedure, and finally, in Section 6.3 we describe a particular implementation of this procedure for $\mathcal{EL}^{+}_{\bot}$.

6.1 Thread-safe Algorithms and Datastructures

Consider the abstract saturation procedure described in Algorithm 1. Suppose there are several independent 'workers' that execute the while loop (lines 4–6) on different processors with the shared Todo and Closure. Will this procedure perform correctly?

The first problem is that with multiple workers, the instruction on line 5 may be executed even if the queue Todo is empty, despite the explicit check on emptiness before that. This is because other workers could have emptied the queue between the time when the worker has checked the queue on emptiness and the time it takes the next element from the queue. This illustrates one of the typical situations when correct serial algorithms stop working correctly in multi-threaded settings. Fortunately, it possible to fix this problem by testing the queue on emptiness when taking the next element from the queue, as shown in Algorithm 5. Here we assume that if Todo is empty, then `Todo.takeNextElement()` returns a distinguished element **null**, which can be used in the condition of the while loop for the emptiness test.

Algorithm 5: Thread-safe while loop of the abstract saturation algorithm

```
while (expression ← Todo.takeNextElement()) ≠ null do        /* close */
    process(expression);
```

A similar problem occurs in the implementation of the method process(expression) on lines 9, 10 of Algorithm 1: it may happen that two workers process the same expression at the same time, both see that it does not occur in Closure, and consequently apply the inference rules two times. Although the outcome of the procedure will still be correct, there will be inefficiency due to the duplicate work. It is likewise possible to avoid this problem by inserting an element in the set and testing if it occurs there at the same time. To this end, we assume that the method Closure.add(expression), in addition to inserting expression into Closure, returns **true** if expression did not occur in Closure and **false** otherwise. It is then easy to see that we can avoid this problem by moving Closure.add(expression) from line 10 to the condition of the if statement on line 9 of Algorithm 1.

The above modifications, however, are not enough to ensure that our algorithm behaves correctly with multiple workers, i.e., that it is *thread-safe*. It is also necessary that all operations with Todo and Closure work as expected even if called from multiple workers at the same time. That is, the datastructures implementing those collections must also be *thread-safe*. For example, when calling Closure.add(expression) from multiple workers for the same expression, this operation should return **true** for at most one worker. Standard implementation for collections may internally use similar control structures like in Algorithm 1, therefore, they are subject to the issues discussed above. It is possible to make sure that Closure and Todo are never accessed from multiple workers using locks. But this largely defeats the purpose of parallelization since the workers would have to wait for each other in order to progress. Truly thread-safe datastructures that allow multiple workers to perform operations concurrently are difficult to implement, and even if implemented, may exhibit certain overheads when compared to conventional datastructures, which can reduce or even cancel out the potential gains. Existing programming libraries, however, typically include efficient thread-safe implementations of common datastructures used in concurrent programming, such as queues. In the next section, we demonstrate how to modify Algorithm 1 so that only Todo is required to be thread-safe, but not necessarily Closure.

6.2 The Abstract Concurrent Saturation Procedure

In this section we describe a concurrent modification of the saturation algorithm for computing the closure, which uses just concurrent queues and Booleans with atomic 'compare and swap' operations. The main idea is to distribute expressions according to 'contexts' in which the expressions can be used as premises of inference rules, and which can be processed independently by the workers. To this end, we assume that there is a finite set of *contexts* and a function getContexts(expression) assigning to every expression a subset of contexts such that, whenever an inference between several expressions is possible, at least one common context is assigned to all of these expressions. In Section 6.3 we present a concrete context assignment satisfying this requirement for the $\mathcal{EL}_\bot^+$ rules in Fig. 3.

Intuitively, the concurrent saturation procedure works as follows. When an expression is derived, it is 'sent' to all contexts assigned to this expression. Whenever a context receives an expression, it is 'activated' to be processed by some worker. Whenever a worker becomes

Algorithm 6: The abstract concurrent saturation procedure

```
1  while (expression ← Input.takeNextElement()) ≠ null do
2      for $c$ ∈ getContexts(expression) do
3          $c$.enqueue(expression);
4  while ($c$ ← activeContexts.takeNextElement()) ≠ null do
5      while (expression ← $c$.Todo.takeNextElement()) ≠ null do
6          $c$.process(expression);
7      deactivate($c$);
8  $c$.process(expression) :
9      if $c$.Closure.add(expression) then
10         for inference ∈ inferences(expression, $c$.Closure) do
11             for $c'$ ∈ getContexts(inference.conclusion) do
12                 $c'$.enqueue(inference.conclusion)
13 $c$.enqueue(expression) :
14     $c$.Todo.add(expression);
15     activate($c$);
16 activate($c$) :
17     if $c$.isActive.compareAndSet(false, true) then
18         activeContexts.add($c$);
19 deactivate($c$) :
20     $c$.isActive ← false;
21     if $c$.Todo ≠ ∅ then activate($c$);
```

available, it takes the next activated context and applies all inference rules, but only to the expressions which were sent to this context. The conclusions of such inferences can be sent again to multiple contexts and this process repeats.

This idea is realized in Algorithm 6. We assume that every context c has a separate queue c.Todo and a separate set c.Closure, whose purpose is similar to those of Todo and Closure from Algorithm 1, but they only contain expressions to which c is assigned. In addition, the procedure maintains a queue activeContexts to keep all contexts c for which c.Todo is not empty. Queuing of expressions for processing is now done by calling, for every context c assigned to the given expression, the function c.enqueue(expression) (lines 13–15) which inserts expression into c.Todo and *activates* c. Activation of a context c should result in context c being added to activeContexts (see line 18, ignore line 17 for now). Each activated context is then repeatedly processed by workers in the main loop of Algorithm 6 (lines 4–7) similarly as in the main loop of Algorithm 1, except that the inferences are performed with the elements of context-local Todo and Closure. Since by our assumption, expressions can participate in inferences only if they are assigned to some common context, no inferences will be lost in this way. Note that the set of contexts and context assignment are always fixed and do not change during the saturation (but some contexts might never be activated).

In Algorithm 6 we assume that the queues Input, c.Todo for every c, and activeContexts are thread-safe, but we do not require thread-safety for any c.Closure. Instead, we ensure that each c.Closure is never accessed by more than one worker at a time. Note that c.Closure can only be accessed from within the method c.process(expression), which is only called after c is taken from the activeContexts queue in the main loop of the algorithm (lines 4–7). Therefore, it is sufficient to ensure that c can never be inserted into activeContexts if it already occurs there or the method c.process(expression) is being called by some worker. For this purpose, we use a Boolean flag c.isActive, which is set to **true** every time c is inserted into activeContexts and set back to **false** after c has been processed by some worker.

Take a look at the instruction on line 17 of method activate(c). There are two things that happen in this instruction. First, it is checked if the value of c.isActive is **false**, and if this is the case, it is set to **true**. Second, if the value of c.isActive has changed (from **false** to **true**), then the context c is inserted into activeContexts. This mechanism makes sure that (i) if c occurs in activeContexts then c.isActive is **true**, and (ii) c is inserted into activeContexts only if c.isActive was **false**. This way, c cannot be inserted into activeContexts if it already occurs there or being processed by a worker (during which c.isActive remains **true**).

It is, however, essential that the flag c.isActive is compared and changed in one instruction. Take a look at Algorithm 7, which provides a seemingly equivalent implementation of the method activate(c). This implementation is no-longer thread safe. Indeed, if two

Algorithm 7: Non-thread-safe activation of contexts

```
activate($c$):
    if $c$.isActive = false then
        c.isActive ← true;
        activeContexts.add($c$);
```

workers call activate(c) for some c at the same time, they can both see that c.isActive is **false**, set it to **true**, and insert c into activeContexts two times, after which c may be simultaneously processed by two different workers. This cannot happen with the implementation in Algorithm 6: if several workers call isActive.compareAndSet(**false**, **true**) at the same time, only for one of them this method can return **true**.

Note that the implementation of the queues c.Todo and activeContexts must still be thread-safe since several workers can insert elements into these queues at the same time during the calls of the function c.enqueue(expression). Also note that activation of a context c on line 15 of method c.enqueue(expression) should happen after inserting expression into c.Todo, not before that (that is, one cannot swap lines 14 and 15). Otherwise c may be inserted into activeContexts, processed by another worker, and deactivated before expression is inserted into c.Todo, and we end up with a context whose Todo queue is not empty, but it is not activated. Also note that line 21 is necessary because after deactivate(c) is called and before c.isActive is set to **false**, some other worker could insert an expression into c.Todo, so c must be reactivated in this case. These issues demonstrate that designing thread-safe algorithms is a difficult task and such algorithms are highly fragile.

6.3 The Concurrent Saturation Procedure for $\mathcal{EL}^+_\bot$

In this section we discuss how to turn the saturation procedure for $\mathcal{EL}^+_\bot$ presented in Section 4.2.3 into a concurrent one using Algorithm 6. First, we need to find a suitable context assignment that satisfies the requirement for the inference rules in Fig. 3 (that the premises of every rule should be assigned to at least one common context). A simple solution would be to use the inference rules themselves as contexts and assign to every expression the set of inference rules in which the expression can participate. This strategy, however, provides only for as many contexts as there are inference rules, so it may not be very effective with

Table 3 Representation of $\mathcal{EL}_\bot^+$ expressions within contexts

Expression	Context c	Representation in c.Todo	Representation in c.Closure
$\mathsf{init}(C)$	C	Init	C.Init $=$ **true**
$C \sqsubseteq D$	C	$\mathsf{Sub}(D)$	$D \in C$.subs
$E \overset{R}{\to} C$	C	$\mathsf{BackLink}(R,E)$	$\langle R,E \rangle \in C$.backLinks
	E	$\mathsf{ForwLink}(R,C)$	$\langle R,C \rangle \in E$.forwLinks

many workers. To find a better solution, note that all premises of the rules in Fig. 3 always have a common concept denoted as C. So, instead of assigning expressions to rules, we can assign expressions to the corresponding concepts. This gives us a much finer granularity for concurrent processing: there can be as many contexts as there are concepts in the input ontology. This should translate into a better scalability even with a large number of workers.

This context assignment determined by the rules in Fig. 3 is thus as follows: expressions of the form $\mathsf{init}(C)$, $C \sqsubseteq D$, or $E \overset{R}{\to} C$ are assigned to the context C. Additionally, $E \overset{R}{\to} C$ is also assigned to the context E because the expression may be used as the second premise of rule $\mathsf{R}_\circ$. Note that this assignment of contexts is optimal in the sense that every inference is possible in exactly one context (since the context C is uniquely determined by the inference), so the overall number of inferences performed by the workers does not increase.

Since the contexts c under our assignment are parts of expressions, it is possible to further 'compress' the representations of elements in c.Todo and c.Closure by removing the arguments that are uniquely determined by the context c as presented in Table 3. The context-local version of function `process`(expression) from Algorithm 2 that uses this context assignment and representation is given in Algorithm 8.

Example 11 In this example we show how the inferences from Example 4 are transformed when applying the concurrent saturation procedure in Algorithm 6. In Table 4 we have listed all conclusions (23)–(37) in their Todo representation in the order they are derived in the respective contexts. The rules applied to these conclusions by Algorithm 8 are listed in the last column of the table. The rules indicate in which contexts they are applied to the current expression (assuming all the previous expressions have been processed), and which table entries from Examples 6 and 8 that were precomputed during the phases 1 and 2 of the saturation procedure (see Section 4.2) are used in the inferences. The conclusion of every inference appears in the respective context below the current line, unless it has already been derived (we do not list duplicate conclusions because no inference applies to them). Contexts are activated when the first unprocessed conclusion is derived in the context and deactivated when no unprocessed conclusions are left. For example, context A is activated when Init is produced, deactivated after $\mathsf{ForwLink}(R, C \sqcap D)$ is processed, activated again when $\mathsf{Sub}(\exists S.D)$ is derived, and, finally, deactivated after $\mathsf{Sub}(B)$ is processed. Note that the contexts A and D can be active at the same time, and thus can be processed in parallel.

7 System Overview

A practical implementation of the reasoning methods that we explained above is provided in the form of the ontology reasoner ELK, which is described in this section. ELK is a Java-based system to reason with OWL ontologies under Direct Semantics, which can be viewed

Algorithm 8: C.process(expression)

```
$C$.process(Init) :
    if $C$.Init.compareAndSet(false,true) then
        $C$.enqueue(new Sub($C$));                                        // rule $R_0$
        if top.negOccurs $> 0$ then
            $C$.enqueue(new Sub(top));                                    // rule $R_\top$

$C$.process(Sub($D$)) :
    if $C$.subs.add($D$) then
        if $D =$ bottom then
            for each $E, R$ with $C$.backLinks($E, R$) do
                $E$.enqueue(add(new Sub(bottom)));                        // rule $R_\bot$
        if $D$ instanceOf IdxConjunction then
            $C$.enqueue(new Sub($D$.firstConj));
            $C$.enqueue(new Sub($D$.secondConj));                         // rule $R_\sqcap^-$
        for each $D_2, E$ with $C$.subs($D_2$) and negConjs($D, D_2, E$) do
            $C$.enqueue(new Sub($E$));                                    // rule $R_\sqcap^+$
        for each $D_1, E$ with $C$.subs($D_1$) and negConjs($D_1, D, E$) do
            $C$.enqueue(new Sub($E$));                                    // rule $R_\sqcap^+$
        if $D$ instanceOf IdxExistential then
            $D$.filler.enqueue(new BackLink($D$.role, $C$));
            $C$.enqueue(new ForwLink($D$.role, $D$.filler)) ;             // rule $R_\exists^-$
        for each $E, F, R, S$ with $C$.backLinks($R, E$) and negExists($S, D, F$) and hier($R, S$) do
            $E$.enqueue(new Sub($F$));                                    // rule $R_\exists^+$
        for each $E$ with conclncs($D, E$) do
            $C$.enqueue(new Sub($E$));                                    // rule $R_\sqsubseteq$

$C$.process(BackLink($R, E$)) :
    if $C$.backLinks.add($\langle R, E \rangle$) then
        if $C$.subs.contains(bottom) then
            $E$.enqueue(new Sub(bottom));                                 // rule $R_\bot$
        for each $D, F, S$ with $C$.subs($D$) and negExists($S, D, F$) and hier($R, S$) do
            $E$.enqueue(new Sub($F$));                                    // rule $R_\exists^+$
        for each $D, R_2, S_1, S_2, S$ with $C$.forwLinks($R_2, D$) and roleComps($S_1, S_2, S$) and
        hier($R, S_1$) and hier($R_2, S_2$) do
            $D$.enqueue(new BackLink($S, E$));
            $E$.enqueue(new ForwLink($S, D$));                            // rule $R_\circ$
        $C$.enqueue(Init);                                                // rule $R_\rightsquigarrow$

$C$.process(ForwLink($R, D$)) :
    if $C$.forwLinks.add($\langle R, D \rangle$) then
        for each $E, R_1, S_1, S_2, S$ with $C$.backLinks($R_1, E$) and roleComps($S_1, S_2, S$) and
        hier($R_1, S_1$) and hier($R, S_2$) do
            $D$.enqueue(new BackLink($S, E$));
            $E$.enqueue(new ForwLink($S, D$));                            // rule $R_\circ$
```

Table 4 The rule applications from Example 4 using the concurrent saturation procedure

Context A	Context $C \sqcap D$	Context D	Context : Rule[Precomputed Table Entries]
Init			$A : \mathsf{R}_0$
$\mathsf{Sub}(A)$			$A : \mathsf{R}_\sqsubseteq[\mathsf{conclncs}(A, \exists R.(C \sqcap D))]$
$\mathsf{Sub}(\exists R.(C \sqcap D))$			$A : \mathsf{R}_\exists^-$
$\mathsf{ForwLink}(R, C \sqcap D)$	$\mathsf{BackLink}(R, A)$		$C \sqcap D : \mathsf{R}_\rightsquigarrow$
	Init		$C \sqcap D : \mathsf{R}_0$
	$\mathsf{Sub}(C \sqcap D)$		$C \sqcap D : \mathsf{R}_\sqcap^-$
	$\mathsf{Sub}(C)$		
	$\mathsf{Sub}(D)$		$C \sqcap D : \mathsf{R}_\exists^+[\mathsf{negExists}(S, D, \exists S.D), \mathsf{hier}(R,S)]$
$\mathsf{Sub}(\exists S.D)$			$A : \mathsf{R}_\exists^-,\ \mathsf{R}_\sqcap^+[\mathsf{negConjs}(A, \exists S.D, A \sqcap \exists S.D)],$ $\mathsf{R}_\sqsubseteq[\mathsf{conclncs}(\exists S.D, C)]$
$\mathsf{ForwLink}(S, D)$		$\mathsf{BackLink}(S, A)$	$D : \mathsf{R}_\rightsquigarrow$
$\mathsf{Sub}(A \sqcap \exists S.D)$		Init	$A : \mathsf{R}_\sqcap^-, \mathsf{R}_\sqsubseteq[\mathsf{conclncs}(A \sqcap \exists S.D, B)] \mid D : \mathsf{R}_0$
$\mathsf{Sub}(C)$		$\mathsf{Sub}(D)$	$D : \mathsf{R}_\exists^+[\mathsf{negExists}(S, D, \exists S.D), \mathsf{hier}(S,S)]$
$\mathsf{Sub}(B)$			$A : \mathsf{R}_\sqsubseteq[\mathsf{conclncs}(B, A \sqcap \exists S.D)]$

as a slightly richer syntax for the DL ontologies discussed in this paper. ELK is free and open source, using a commercial-friendly Apache 2 license.[8]

As of this paper, the latest stable release ELK 0.3.2 supports all features of $\mathcal{EL}_\bot^+$, that is, conjunction (ObjectIntersectionOf), existential restriction (ObjectSomeValuesFrom), top (owl:Thing), bottom (owl:Nothing), and complex role inclusions (property chains). This also covers transitive and reflexive properties and disjoint classes, for which OWL provides syntactic shortcuts. Moreover, ELK implements support for ABoxes and ObjectHasValue restrictions as discussed in Appendix A. Finally, ELK provides some preliminary support for datatypes, using a simplified syntactic matching to compare values. The reasoning tasks supported in ELK are ontology consistency checking, TBox classification, and ABox realization.

ELK implements the concurrent saturation algorithm described in Section 6 and all optimization techniques detailed in Section 5. The number of concurrent workers used by ELK can be configured, where the same number will be used for all reasoning tasks (saturation of roles (properties), saturation of concepts (classes), transitive reduction). The default is to use the number of cores reported by the operating system; this usually includes virtual cores. In addition, ELK generally uses one parallel worker to generate the ontology index (Section 4.2.1) while loading an ontology.

ELK is a flexible system that can be used in a variety of configurations. This is supported by a modular program structure that is organized using the Apache Maven[9] build manager for Java. Maven can be used to automatically download, configure, and build ELK and its dependencies, but there are also pre-built packages for the most common configurations. The modular structure also separates the consequence-based reasoning engine from the remaining components, which facilitates extension of the system with new language features.

The main software modules of ELK are shown in Fig. 9. The arrows illustrate the information flow during classification. The two independent entry points are the command-line client and the Protégé plug-in to the left. The former extracts OWL ontologies from files in OWL Functional-Style Syntax (FSS), while the latter communicates with Protégé [57] through the OWL API [44]. All further processing is based on ELK's own representation

[8] Download at http://code.google.com/p/elk-reasoner/

[9] http://maven.apache.org

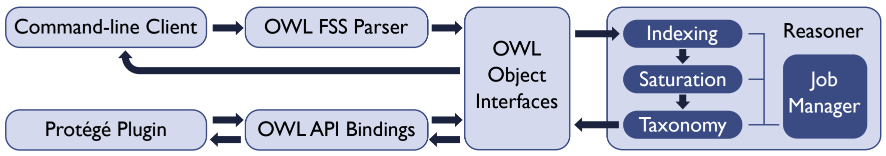

Fig. 9 Main software modules of ELK and information flow during classification

of OWL objects (axioms and expressions) that does not depend on the (more heavyweight) OWL API. The core of ELK is its reasoning module, which was described in Sections 4–6.

ELK is distributed in three pre-built packages, each of which includes the reasoner module. The *standalone client* includes the command-line client and the FSS parser for reading OWL ontologies. The *Protégé plugin* allows ELK to be used as a reasoner in Protégé and compatible tools such as *Snow Owl*.[10] The *OWL API bindings* package allows ELK to be used as a software library that is controlled via the OWL API interfaces.

8 Experimental Evaluation

In this section, we evaluate the classification algorithm and the optimizations presented in this paper on existing ontologies. In Section 8.1, we compare the performance of ELK in its default settings (all optimizations turned on) with other commonly used OWL and OWL EL reasoners. To evaluate the effect of individual optimizations, in Section 8.2 we repeat the experiments with certain optimizations turned off. In Section 8.3 we evaluate the effect of varying the number of concurrent workers. Finally, in Section 8.4 we compare the two transitive reduction algorithms from Section 5.4.

The experiments were executed on a laptop with Intel Core i7-2630QM 2GHz quad-core CPU and 6GB RAM running Microsoft Windows 7. On this architecture, ELK defaults to using 8 concurrent workers in the saturation phase. We ran Java 1.6 with the `-XX:+AggressiveHeap` flag and 4GB of heap space. All figures reported in this paper were obtained as the average over 5 runs of the experiment.

Our test ontology suite contains SNOMED CT obtained from the official January 2012 international release by converting from the native syntax (RF2) to FSS using the supplied converter. Additionally, we used a new experimental version ANATOMY, which remodels the 'body structure' hierarchy of SNOMED CT using role composition axioms. Both of these ontologies are freely available for research and evaluation.[11] We included several versions of OpenGALEN.[12] GALEN7 and GALEN8 were obtained from versions 7 and 8 respectively by removing all inverse role and functional role axioms, and replacing all data property restrictions with new atomic concepts. The role composition axioms of GALEN7 and GALEN8 do not satisfy the regularity condition imposed by OWL2 and therefore are not proper OWL EL ontologies. Although this is not problematic for ELK, these ontologies are rejected by most OWL reasoners. We have additionally included GALEN-OWL, a proper OWL EL version of GALEN, which is obtained from the CO-ODE version of GALEN[13] by removing inverse role and functional role axioms. This version of the ontology has been used extensively in the past for evaluating reasoners [29,51,53,73,94]. It is similar to GALEN7

[10] http://www.b2international.com/portal/snow-owl
[11] http://www.ihtsdo.org/licensing/
[12] http://www.opengalen.org/sources/sources.html
[13] http://www.co-ode.org/galen/

Table 5 Ontology metrics: number of axioms

	$C \sqsubseteq D$	$C \equiv D$	Disj(C,D)	$R \sqsubseteq S$	$R \equiv S$	Trans(R)	$R_1 \circ R_2 \sqsubseteq S$
Complex:							
SNOMED CT	227,961	66,507	-	11	-	-	1
ANATOMY	17,551	21,831	-	4	-	3	2
GALEN-OWL	25,563	9,968	-	958	-	58	-
GALEN7	27,820	15,270	-	972	14	-	385
GALEN8	53,449	113,622	-	996	14	-	385
GO2	66,216	7,361	6	2	-	2	3
Simple:							
GO1	28,896	-	-	-	-	1	-
ChEBI	67,182	-	-	-	-	2	-
EMAP	13,730	-	-	-	-	-	-
FMA	126,544	-	-	3	-	1	-
Fly Anatomy	19,137	-	61	10	-	3	-
Molecule Role	9,627	-	-	-	-	2	-

Table 6 Ontology metrics: number of concepts, roles, and constructors by occurrence polarities

	A	pos. $\sqcap$	neg. $\sqcap$	pos. $\exists$	neg. $\exists$	R
Complex:						
SNOMED CT	294,469	251,428	140,554	105,373	75,666	62
ANATOMY	37,757	49,092	4,729	25,880	21,387	10
GALEN-OWL	23,136	13,006	12,542	14,115	7,549	950
GALEN7	28,482	13,079	12,982	15,105	7,973	964
GALEN8	128,483	141,592	140,542	106,065	93,241	988
GO2	36,215	7,363	7,363	10,157	6,581	7
Simple:						
GO1	20,465	-	-	1,796	-	1
ChEBI	31,190	-	-	14,053	-	9
EMAP	13,731	-	-	4,821	-	1
FMA	80,469	-	-	13,691	-	15
Fly Anatomy	7,797	-	-	2,558	-	40
Molecule Role	9,217	-	-	2,238	-	4

but its only role compositions are transitivity axioms. We also used two versions of the Gene Ontology[14] which we call GO1 and GO2. The older GO1, published in 2006, has been used in many performance experiments [14,29,34,51,73,94,104]. GO2 is the version of March 2012 and uses significantly more features than GO1, including negative occurrences of conjunctions and existential restrictions, and even a few disjointness axioms. To obtain further test data, we selected some of the largest ontologies listed at the OBO Foundry [97] and the Ontobee [111] websites that were in OWL EL but were not just plain taxonomies, i.e., included some non-atomic concepts. This gave us the Chemical Entities of Biological Interest (ChEBI), the e-Mouse Atlas Project (EMAP), the Foundational Model of Anatomy (FMA), the Fly Anatomy, and the Molecule Role ontology. All ontologies that we are allowed to publish can be downloaded from the ELK website.[15]

Tables 5 and 6 show various statistics about the ontologies from our benchmark suite. Table 5 shows the number of various axiom types; the only logical axiom not mentioned in the table is one reflexive role axiom in ANATOMY. Table 6 shows the number of atomic concepts, roles, and the number of positive and negative occurrences of conjunctions and existentials. It turns out that many of the smaller ontologies in our suite contain only concept

[14] http://www.geneontology.org
[15] http://code.google.com/p/elk-reasoner/wiki/TestOntologies

inclusion axioms of the very simple form $A \sqsubseteq B$ and $A \sqsubseteq \exists R.B$, where A and B are atomic concepts. We will refer to these ontologies as *simple* and to other as *complex* as indicated in Tables 5 and 6. It is interesting to note that the simple ontologies can be fully classified just by computing the transitive closure of the told subsumptions $A \sqsubseteq B$ ignoring the remaining axioms. To the best of our knowledge, no reasoner currently takes advantage of this fact. ELK also applies rule $\mathsf{R}^+_\exists$ to the positive existentials in these ontologies even though, due to lack of negative existentials, the resulting links can never participate in rule $\mathsf{R}^-_\exists$.

All of our experiments are focused on terminological reasoning, which is currently the most common reasoning problem used in applications involving $\mathcal{EL}$ ontologies [33,39,48, 81,85]. Although the OWL EL standard comprises many features, such as assertions, nominals, and datatypes, these are difficult to find in existing OWL EL ontologies. One of the reason is that many ontologies were not developed in OWL from the beginning, but have been converted to OWL from other formats, such as OBO [28], Grail [86], or frame-like languages, which did not have those features. In our previous experiments with nominals [56], we had to resort to synthetically generated data, but, arguably, such experiments are of a limited value. For the same reason, we also do not evaluate the optimized reasoning with disjointness axioms described in Section 5.3. For a quick (synthetic) evaluation, we modified SNOMED CT by declaring all leaf concepts (i.e., concepts that do not subsume other atomic concepts) disjoint, leading to a disjointness axiom with about 200,000 concepts. This did not lead to any significant difference in ELK's classification time compared to the original ontology.

8.1 Performance Comparison with Other Reasoners

We compared ELK 0.3.2 to the specialized OWL EL reasoners CEL 1.1.2 [14], jcel 0.18.0 [72], the REL reasoner from TrOWL 1.2 [102], and Snorocket 2.0.5 [66], to general OWL reasoners FaCT++ 1.6.0 [103], HermiT 1.3.6 [76], JFact 0.9,[16] Pellet 2.3.0 [96], and RacerPro 2.0 build 20121209 [36], and to experimental consequence-based reasoners CB r.12 [51] and ConDOR r.12 [95]. We did not perform experiments with Cheetah [94] because by the time of writing, this reasoner was not released. We ran all reasoners in their default settings.

CEL, jcel, REL, and Snorocket are typical OWL EL systems that implement completion-based procedures [9]. CB and ConDOR are prototype implementations of consequence-based algorithms for logics that are more expressive than $\mathcal{EL}$. FaCT++, JFact, Pellet, and RacerPro use tableau algorithms, and HermiT is based on a hypertableau calculus. The general OWL reasoners may also use other more efficient reasoning methods when applied to $\mathcal{EL}$ ontologies. Section 9 provides further details, and also discusses various other systems that we have not included in this evaluation. Recent versions of Snorocket implement a context-based concurrent procedure which is similar to ours; like ELK, on our architecture Snorocket defaults to using 8 concurrent workers. The remaining reasoners do not take advantage of concurrency.

Due to the technical differences between the systems, we have used two different experimental setups for our evaluation. Most reasoners could be evaluated using the standard interface of the OWL API [44], which allows us to access the reasoners uniformly and facilitates fair comparison. For the case of CB, CEL, ConDOR, and RacerPro, this general setup was not appropriate, for a variety of reasons as explained below. The results obtained in

[16] http://jfact.sourceforge.net/

Table 7 Classification time in seconds, measured using the OWL API

	ELK	jcel	REL	Snorocket	FaCT++	HermiT	JFact	Pellet
SNOMED CT	5.1	651.4	116.2	25.8	425.2	time	time	mem
ANATOMY	4.0	180.0	stack	27.8	N/A	N/A	N/A	N/A
GALEN-OWL	1.2	30.0	27.8	2.9	time	time	time	mem
GALEN7	1.5	57.9	stack	7.9	N/A	N/A	N/A	N/A
GALEN8	5.8	time	stack	mem	N/A	N/A	N/A	N/A
GO2	1.1	8.2	11.3	2.5	time	41.2	time	65.7
GO1	0.5	2.2	0.9	1.1	6.8	2.6	10.0	2.5
ChEBI	0.7	7.6	3.2	1.9	3.5	12.5	7.7	exc
EMAP	0.3	0.9	0.5	0.6	20.0	2.0	37.7	0.8
FMA	1.0	16.4	8.8	7.1	5.6	20.7	13.2	736.4
Fly Anatomy	0.4	2.2	1.0	0.8	0.7	1.8	2.8	23.1
Molecule Role	0.3	1.0	0.4	0.6	5.4	1.4	9.4	0.9

Table 8 Loading + classification time in seconds, measured using the OWL API

	ELK	jcel	REL	Snorocket	FaCT++	HermiT	JFact	Pellet
SNOMED CT	9.3	674.3	126.2	38.2	431.3	time	time	mem
ANATOMY	5.0	182.3	stack	29.2	N/A	N/A	N/A	N/A
GALEN-OWL	2.0	32.3	29.2	4.3	time	time	time	mem
GALEN7	2.3	60.2	stack	9.2	N/A	N/A	N/A	N/A
GALEN8	11.1	time	stack	mem	N/A	N/A	N/A	N/A
GO2	2.1	9.9	12.1	3.8	time	44.0	time	67.9
GO1	1.0	3.0	1.2	1.7	7.3	3.7	10.3	3.6
ChEBI	1.3	8.8	3.6	2.7	4.2	13.9	8.2	exc
EMAP	1.0	1.6	0.7	1.0	20.4	3.0	38.0	1.8
FMA	2.2	18.6	9.3	8.4	7.4	23.1	14.0	741.8
Fly Anatomy	0.8	2.9	1.3	1.3	1.1	2.8	3.2	24.1
Molecule Role	0.6	1.6	0.5	1.0	5.7	2.1	9.6	1.5

these cases can still be useful indicators of general performance, but some caution is needed when using them to compare systems.

In the first experimental setup, we parsed and loaded the ontologies using the OWL API 3.4. Table 7 shows the wall-clock time each reasoner spent executing the classification method `precomputeInferences(CLASS_HIERARCHY)`. Note, however, that a reasoner may perform certain computations already during ontology loading before calling the classification method; these typically include normalization and indexing of axioms. For this reason, in Table 8 we also show the overall wall-clock time for loading and classification. Possible failures for a reasoner are *time* (no result after 30min), *mem* (out-of-memory error), *stack* (stack overflow), *N/A* (reasoner rejects the ontology due to non-regular role compositions), and *exc* (program error).

In our second experimental setup, we measured classification times using a specific method for each reasoner. We ran CB as a plugin in Protégé 4.2, ConDOR and CEL from the command line, and RacerPro using its client RacerPorter. For CB, ConDOR, and RacerPro, these are the setups suggested by the developers for most accurate evaluation. CEL requires a Unix-like operating system; we used Linux Mint 13 on the same hardware as in all other experiments. CB, ConDOR, and RacerPro were evaluated on the same Microsoft Windows 7 platform as all other systems. Table 9 shows the classification times as reported by the reasoners. The only form of role composition supported by CB and ConDOR is transitivity, hence they are not applicable to any complex ontology apart from GALEN-OWL. However, the single role composition in SNOMED CT is redundant in the sense that rule

Table 9 Classification time in seconds, reported by the reasoners not supporting OWL API

	CB	ConDOR	CEL	RacerPro
SNOMED CT	36.5	43.8	772.3	778.5
ANATOMY	N/A	N/A	144.0	N/A
GALEN-OWL	3.7	4.4	103.9	time
GALEN7	N/A	N/A	88.0	N/A
GALEN8	N/A	N/A	time	N/A
GO2	N/A	N/A	24.0	mem
GO1	0.5	0.4	0.6	4.2
ChEBI	2.1	2.2	81.8	stack
EMAP	0.2	0.1	0.1	13.3
FMA	3.1	2.0	216.8	22.7
Fly Anatomy	0.3	0.2	1.5	mem
Molecule Role	0.2	0.1	0.1	3.5

$\mathsf{R}_\circ$ is never applied during classification, so we decided to measure the running times of CB and ConDOR on SNOMED CT even though they discard this role composition.

Overall, the results of the evaluation show that ELK compares favorably with the other reasoners. While many reasoners in our comparison show similar running times on the simple ontologies, ELK has a significant advantage on the complex ontologies. In particular, ELK is the only reasoner that can classify GALEN8. It can load and classify SNOMED CT in under 10 seconds. Since ELK can update its index structure incrementally without having to reload the whole ontology, subsequent reclassification of SNOMED CT due to small changes in the ontology is likely to take only about 5 seconds as reported in Table 7. Regarding memory requirements, we can report that in our experiments ELK could classify SNOMED CT with only 2GB of heap space when used through the OWL API, and with as little as 1GB of heap space when used standalone.

8.2 Optimizations of Inference Rules

In our next experiment we evaluated the effect of the optimizations $O_\sqcap$ and $O_\exists$ from Section 5.1, and of the optimization $O_\circ$ from Section 5.2. We excluded the simple ontologies from this experiment: they have no negative occurrences of conjunctions and existential restrictions, so $O_\sqcap$ and $O_\exists$ do not apply, and, although some of the simple ontologies contain transitive roles, there are no subrole relationships between transitive roles in these ontologies, so $O_\circ$ does not apply either.

We evaluated five configurations of the classification algorithm: with none of the three optimizations $O_\sqcap$, $O_\exists$, and $O_\circ$, with one of these optimizations turned on at a time, and with all the three optimizations together (the default setting). We measured the overall classification time with one concurrent worker, the number of derived axioms including multiplicity, and the number of uniquely derived axioms. The results are shown in Table 10.

First, observe that on SNOMED CT no link $C \overset{R}{\to} D$ is derived more than once. This is because, even though SNOMED CT contains one role composition axiom, rule $R_\circ$ is never applied on this ontology. The links are therefore derived only by rule $\mathsf{R}_\exists^-$, which can never produce the same link twice. Next, we discuss each individual optimization in turn.

The optimization $O_\sqcap$ avoids the decomposition of $C \sqsubseteq D_1 \sqcap D_2$ into $C \sqsubseteq D_i$ for $i = 1,2$ in case the former subsumption has been obtained by the composition of the latter two. Thus, the optimization can decrease the multiplicity but not the number of unique subsumptions, and it has no effect on links at all. Furthermore, the optimization makes the multiplicity

Table 10 Classification time in seconds (1 working thread) and number of derived axioms

	time	speedup	derived $C \sqsubseteq D$	unique $C \sqsubseteq D$	derived $C \overset{R}{\rightsquigarrow} D$	unique $C \overset{R}{\rightsquigarrow} D$
SNOMED CT						
no optimization	26.31	1.00	47,435,318	13,840,227	3,969,744	3,969,744
with $O_\sqcap$	25.48	1.03	41,770,050	13,840,227	3,969,744	3,969,744
with $O_\exists$	19.75	1.33	28,438,072	13,840,227	984,775	984,775
with $O_\circ$	26.37	1.00	47,435,318	13,840,227	3,969,744	3,969,744
with $O_\sqcap$, $O_\exists$, $O_\circ$	18.71	1.41	22,772,804	13,840,227	984,775	984,775
ANATOMY						
no optimization	28.63	1.00	16,529,447	3,618,582	97,927,757	2,515,236
with $O_\sqcap$	29.01	0.99	16,495,539	3,618,582	97,927,757	2,515,236
with $O_\exists$	16.38	1.75	12,045,017	3,618,582	46,301,813	1,511,399
with $O_\circ$	21.83	1.31	16,529,447	3,618,582	62,674,413	2,515,236
with $O_\sqcap$, $O_\exists$, $O_\circ$	11.70	2.45	12,011,440	3,618,582	25,295,665	1,511,418
GALEN-OWL						
no optimization	3.31	1.00	2,860,224	1,147,483	759,473	405,955
with $O_\sqcap$	3.26	1.01	2,340,868	1,147,483	759,473	405,955
with $O_\exists$	2.65	1.25	2,182,677	1,147,483	251,312	177,185
with $O_\circ$	3.42	0.97	2,856,130	1,147,483	713,681	399,956
with $O_\sqcap$, $O_\exists$, $O_\circ$	2.53	1.31	1,644,288	1,147,483	208,749	167,623
GALEN7						
no optimization	6.84	1.00	7,277,608	2,058,039	5,045,114	941,723
with $O_\sqcap$	6.74	1.01	6,410,540	2,058,039	5,045,114	941,723
with $O_\exists$	4.67	1.46	5,240,770	2,058,039	1,521,973	374,043
with $O_\circ$	6.21	1.10	6,909,517	2,058,039	3,043,228	837,402
with $O_\sqcap$, $O_\exists$, $O_\circ$	4.22	1.62	3,976,718	2,058,039	572,405	286,239
GALEN8						
no optimization	50.39	1.00	69,138,922	14,248,354	46,241,197	7,443,869
with $O_\sqcap$	48.63	1.04	62,822,068	14,248,354	46,241,197	7,443,869
with $O_\exists$	25.21	2.00	37,267,987	14,248,354	8,871,203	1,922,583
with $O_\circ$	43.83	1.15	63,882,676	14,248,354	26,741,691	6,627,105
with $O_\sqcap$, $O_\exists$, $O_\circ$	20.65	2.44	26,111,096	14,248,354	2,749,394	1,389,498
GO2						
no optimization	2.02	1.00	1,992,627	718,866	315,633	199,001
with $O_\sqcap$	2.01	1.01	1,990,869	718,866	315,633	199,001
with $O_\exists$	2.03	1.00	1,983,811	718,866	291,833	193,477
with $O_\circ$	2.00	1.01	1,992,627	718,866	315,381	199,001
with $O_\sqcap$, $O_\exists$, $O_\circ$	1.96	1.03	1,982,053	718,866	291,599	193,477

of subsumptions sensitive to the order of rule applications: the decomposition of $C \sqsubseteq D_1 \sqcap D_2$ is avoided only if the subsumption is derived by rule $\mathsf{R}^+_\sqcap$ before it is derived by any other rule. The optimization decreases the multiplicity of subsumptions on each ontology in this experiment, albeit for ANATOMY and GO2 the difference is small. In all cases, the differences in classification times were only marginal.

The optimization $O_\exists$ avoids the decomposition of $C \sqsubseteq \exists R.D$ into $\mathsf{init}(D)$ and $C \overset{R}{\rightsquigarrow} D$ in case the first subsumption has been obtained by composition, but in this case it is possible that the avoided conclusions will not be derived by the algorithm at all. Since the optimization can even avoid initialization of concepts, it can decrease all the four numbers shown in Table 10; furthermore, it makes all the four numbers sensitive to the order of rule applications. Even though for the classification task each atomic concept is already initialized on input, the optimization can still avoid initialization of complex concepts in existential restrictions. We have, however, not observed this on any of the ontologies in this experiment, which is why we have obtained the same number of uniquely derived subsumptions both with and without $O_\exists$. On the other hand, the optimization has substantially reduced the remaining three numbers in Table 10 on all the test ontologies apart from GO2, with a speedup of 1.33 on SNOMED CT, 1.75 on ANATOMY, 1.25 on GALEN-OWL, 1.46 on GALEN7, and as much as 2.00 on GALEN8.

The optimization $O_\circ$ avoids the derivation of some links by rule $\mathsf{R}_\circ$ without affecting the set of subsumptions that are derivable by the algorithm. Therefore, the optimization can decrease the multiplicity and the number of unique links, and then, due to $\mathsf{R}_\exists^+$, also the multiplicity of (but not the number of unique) subsumptions. Indeed, the optimization decreases all these three numbers on all the versions of GALEN, with some improvement in classification times for GALEN7 and GALEN8. Since rule $\mathsf{R}_\circ$ is never applicable on SNOMED CT, the optimization has no effect on this ontology. Finally, for ANATOMY and GO2 we only see a decrease in the multiplicity of links: this is negligible for GO2 but considerable for ANATOMY where the speedup reaches 1.31. Unlike the previous two optimizations, $\mathsf{R}_\circ$ is not sensitive to the order of rule applications.

Finally, we discuss the case of using all three optimizations together. Since there is no interaction between $O_\sqcap$ and the remaining two optimizations, adding $O_\sqcap$ to $O_\exists$ and/or $O_\circ$ results in exactly the same reduction in the multiplicity of subsumptions as with $O_\sqcap$ alone. More interestingly, $O_\exists$ and $O_\circ$ optimize the derivation of links in two different ways, and our experiments show their combined effect can be considerably larger than the effect of either of the two optimizations alone.

Although all the three optimizations had only limited effect on GO2, they proved to be effective on the remaining ontologies, altogether speeding up classification by a factor of 1.41 on SNOMED CT and as much as 2.45 on ANATOMY and GALEN8. Out of the three optimizations considered in this section, $O_\exists$ appears to be the most useful one, while $O_\sqcap$ does not seem to be very significant in practice. On the other hand, it is trivial to include $O_\sqcap$ if one already implements $O_\exists$. The last optimization $O_\circ$ is effective only on ontologies that have subrole relations between roles occurring in role compositions, such as ANATOMY and the variants of GALEN in our experiments.

8.3 Concurrency

Next, we evaluated the effect of increasing the number of concurrent workers in ELK. Since the machine on which we performed the experiments has 4 physical cores which, due to hyper-threading, appear to the operating system as 8 virtual cores, we experimented with up to 8 concurrent workers. The measured classification times are shown in Table 11.[17]

The results show that increasing the number of workers improves the performance of ELK, and that the improvement is more pronounced on the largest ontologies: while ELK achieves a speedup for 8 workers by a factor of 3.83 on SNOMED CT and 3.40 on GALEN8, the speedups are below 2 on many of the smaller ontologies. To further test the hypothesis that the speedup improves with increasing the size of an ontology, we repeated this experiment on the union of all the simple ontologies. As shown in the last row of Table 11 under the name UNION, this resulted in a speedup by a factor of 2.45 which is considerably higher than for any of the individual ontologies.

Our experiments confirm that concurrent processing can offer improvements for ontology classification on common computing hardware. On the other hand, the experiments demonstrate that the improvement factor is far from linear, and that it appears to be higher

[17] For a fair comparison with other reasoners, we ran ELK in the experiments in Section 8.1 through OWL API. In the remaining experiments, however, we accessed it directly using its own interfaces. This explains the slight difference between the running times in the last column of Table 11 and those in Table 7.

Table 11 Classification time in seconds and relative speedup for increasing number of concurrent workers

		workers							
		1	2	3	4	5	6	7	8
SNOMED CT	time	18.62	10.07	7.37	6.35	5.76	5.49	5.09	4.85
	speedup	1.00	1.85	2.53	2.93	3.23	3.39	3.66	3.84
ANATOMY	time	11.58	7.27	5.51	4.63	4.34	4.03	3.80	3.64
	speedup	1.00	1.59	2.10	2.50	2.67	2.88	3.04	3.18
GALEN-OWL	time	2.49	1.64	1.32	1.27	1.27	1.28	1.25	1.23
	speedup	1.00	1.51	1.88	1.95	1.96	1.94	1.98	2.02
GALEN7	time	4.12	2.57	2.05	1.85	1.74	1.68	1.60	1.67
	speedup	1.00	1.60	2.01	2.23	2.36	2.45	2.58	2.46
GALEN8	time	20.56	12.73	9.21	7.67	7.13	6.71	6.32	6.06
	speedup	1.00	1.62	2.23	2.68	2.88	3.07	3.26	3.40
GO2	time	1.97	1.21	1.05	1.13	1.14	1.11	1.16	1.14
	speedup	1.00	1.63	1.88	1.74	1.73	1.76	1.70	1.72
GO1	time	0.78	0.53	0.52	0.54	0.54	0.54	0.52	0.56
	speedup	1.00	1.47	1.48	1.44	1.44	1.43	1.50	1.38
ChEBI	time	1.50	0.96	0.78	0.79	0.78	0.78	0.80	0.80
	speedup	1.00	1.56	1.92	1.91	1.93	1.92	1.88	1.88
EMAP	time	0.68	0.48	0.44	0.45	0.45	0.41	0.42	0.44
	speedup	1.00	1.42	1.57	1.52	1.51	1.67	1.61	1.57
FMA	time	1.72	1.09	0.95	0.89	0.93	0.90	0.94	0.85
	speedup	1.00	1.58	1.81	1.94	1.85	1.91	1.84	2.02
Fly Anatomy	time	0.71	0.52	0.47	0.47	0.50	0.48	0.47	0.49
	speedup	1.00	1.36	1.52	1.49	1.41	1.48	1.49	1.45
Molecule Role	time	0.62	0.46	0.39	0.41	0.38	0.34	0.37	0.36
	speedup	1.00	1.37	1.59	1.52	1.63	1.81	1.70	1.72
UNION	times	2.88	1.69	1.41	1.28	1.24	1.21	1.18	1.17
	speedup	1.00	1.71	2.04	2.25	2.32	2.38	2.45	2.45

on larger ontologies. There can be many causes for this effect, such as dynamic CPU clocking, shared Java memory management and garbage collection, hardware bottlenecks in CPU caches and data transfer, or JIT compilation overheads.

8.4 Transitive Reduction

Finally, we evaluated the difference between the ‘naive’ Algorithm 3 and the ‘optimized’ Algorithm 4 for transitive reduction from Section 5.4. For this experiment, we implemented the two algorithms exactly as shown in Section 5.4 even though the naive algorithm is incorrect in the presence of equivalent concepts. For each of the two algorithms, Table 12 shows the running time and the number of passes through the inner for loop of the algorithm.

The experiments show that the optimized algorithm is about 2–3 times faster than the naive algorithm on SNOMED CT, ANATOMY, GALEN7, and GALEN8. The optimized algorithm always requires substantially fewer passes through the inner loop, with the exception of the EMAP ontology, which entails no non-trivial subsumptions between atomic concepts at all so that the transitive reduction task is trivial. Interestingly, this reduced number of passes does not always translate into the corresponding performance improvement, possibly because the optimized algorithm performs more set additions and removals, which are more expensive than membership checks performed by the ‘naive’ algorithm.

Table 12 Running time in seconds and the number of passes through the inner loop of the two transitive reduction algorithms from Section 5.4

	naive algorithm		optimized algorithm		
	time	passes	time	passes	speedup
SNOMED CT	4.11	35,745,244	1.61	6,723,839	2.55
ANATOMY	1.37	14,674,573	0.43	2,244,702	3.19
GALEN-OWL	0.42	3,012,583	0.26	614,103	1.62
GALEN7	0.84	89,951,79	0.31	1,535,003	2.71
GALEN8	2.92	27,208,529	1.24	4,789,837	2.35
GO2	0.28	2,172,161	0.20	474,042	1.40
GO1	0.24	782,504	0.23	177,966	1.04
ChEBI	0.45	4,322,716	0.22	843,412	2.05
EMAP	0.11	0	0.11	0	1.00
FMA	0.42	3,754,823	0.26	954,998	1.62
Fly Anatomy	0.22	335,522	0.21	78,209	1.05
Molecule Role	0.17	63,083	0.11	13,974	1.55

9 Related Work

We discuss related work for three different aspects of our contribution: OWL EL reasoning (Section 9.1), consequence- and rule-based reasoning (Section 9.2), and concurrent and distributed reasoning (Section 9.3).

9.1 Reasoning in OWL EL and Beyond

Favorable computational properties have long been an important motivation for the study of the $\mathcal{EL}$ family of description logics [6, 12]. Most OWL EL implementations use variations of *completion-based procedures*, which we described in more detail in Section 3.6. First such procedures were proposed for $\mathcal{ELH}$ [22] and $\mathcal{EL}^{++}$ [8]; later works extend this approach to also cover reflexive roles and range restrictions [9], Boolean role constructors [88], and local reflexivity [60].

A number of reasoners have been implemented for the $\mathcal{EL}$ family. The first such system was CEL [13, 14, 15], whose implementation we described in Section 4.4. Various later systems have reimplemented the 'CEL algorithm'[18] in order to provide better compatibility with tools, such as the OWL API and Protégé, or to improve performance for some ontologies, such as SNOMED CT. These systems include Snorocket [66], TrOWL REL [102], and jcel [71]. A prototype reasoner Cheetah [94] was used to investigate the application of linear-time algorithms for propositional Horn logic in $\mathcal{EL}^{+}$ reasoning. As discussed in Section 4.4, the results suggest that, at least for current $\mathcal{EL}$ ontologies, the performance gains of this optimization do not outweigh the implementation overhead. We have arrived at similar conclusions when experimenting with prototype versions of ELK for reasoning with role chains [54] and (unrestricted) nominals [56]. For example, it is hard to come up with examples that would require non-safe use of nominals in OWL EL ontologies. The procedure for safe nominals, as described in Appendix A of this paper, should be, therefore, sufficient in most of the cases.

Other systems have experimented with alternative approaches to reasoning with $\mathcal{EL}$ ontologies. The reasoners DB [29] and OREL [63] explored the use of relational database

[18] http://www.w3.org/2007/OWL/wiki/Implementations

systems in $\mathcal{EL}$ reasoning. While feasible in principle, this approach does not match the performance or scalability of the best in-memory $\mathcal{EL}$ reasoners. Another recent approach shows the applicability of Answer Set Programming engines to OWL EL reasoning [31], using the DReW reasoner [112] to implement a rule-based calculus for OWL EL [60]. The approach aims at providing efficient use of OWL EL ontologies in *dl-programs*, thus enabling a form of rule-ontology integration. Most recently, a prototype implementation for $\mathcal{EL}$ reasoning on embedded devices has been studied [35]. A particular challenge in this context is the very low amount of available memory that allows only very small ontologies to be classified.

Finally, a number of more general-purpose systems provide some dedicated optimizations for (fragments of) OWL EL. FaCT++ [103] reduces the number of subsumption tests for *completely defined concepts*, which frequently occur in GO1 and SNOMED CT [104]. An extension of this optimization with *structural pseudo-model embedding* has been successfully used by RacerPro to classify SNOMED CT [74]. HermiT [76] uses an optimization that can completely avoid subsumption tests for *deterministic ontologies* (including $\mathcal{EL}$) [34]: subsumptions can be just read out of the models produced for concept satisfiability tests. The latest version of Pellet [96] can apparently switch to a specialized procedure when the ontology is within a fragment of OWL EL.[19] HermiT and Pellet, however, were still unable to classify SNOMED CT in our experiments. CB reasoner uses a consequence-based algorithm for Horn-$\mathcal{SHIF}$ [51], which works similarly to the procedure presented in this paper when restricted to $\mathcal{EL}$ (except for concurrency). A similar support for functional and inverse roles (which are outside of OWL EL) has recently been added to jcel [72].

While most works focus on ontology classification and standard reasoning problems, $\mathcal{EL}$-type logics have also been considered for other reasoning tasks, notably conjunctive query answering [58, 64], least common subsumer computation [12, 17], unification [7], and interpolation [79]. These reasoning services have yet to make it into common tools, although some prototype implementations exist.

9.2 Rule-, Consequence-, and Saturation-Based Reasoning

Rules of inference are a versatile approach to automated deduction, and saturation under a set of inference rules is prominently used in several areas. In databases, this is called *materialization* and has applications in data integration, constraint repair, and query answering [1]. In theorem proving, saturation is a key technique for many resolution-based calculi [18]. In production rule systems, similar ideas are applied to *forward-chaining* of rules [32].

The abstract saturation procedure described in Section 4.1 is inspired by the *given clause* approach/*set of support* strategy in theorem proving [109], which is similar to the *semi-naive evaluation* of Datalog queries [1]. Production rule systems typically employ a variant of the *Rete* algorithm for applying rules, which largely avoids iterations over processed facts by creating more complex structures in working memory [32]. Related methods are the linear evaluation strategy for Horn rules studied in Cheetah [94], and ELK's partial join computation described in Section 4.3. These results show that this approach can be useful in OWL reasoning but does not pay off in all cases.

The $\mathcal{EL}$ reasoning procedure described herein is most closely related to *consequence-based procedures*. The first such procedure was described for Horn-$\mathcal{SHIQ}$ ontologies and implemented in the CB reasoner [51]. Later, similar procedures have been formulated for

[19] http://weblog.clarkparsia.com/2009/11/16/pellet-20-release/

Horn-$\mathcal{SROIQ}$ [80], and (non-Horn) $\mathcal{ALCH}$ [95]. The differences between completion-based, consequence-based procedures [8,9,22], and the *proof-theoretic* procedure by Hofmann [41] are discussed in detail in Section 3.6.

Rule-based approaches have also been applied to reasoning in the OWL 2 Profiles [62]. OWL EL was discussed in Section 9.1 before. Another common use of rule-based calculi is instance retrieval in OWL RL [75] and its sublanguages, especially *pD** (a.k.a. OWL-Horst) [47] and RDFS [24]. These calculi include inference rules that are sound only under the *RDF-Based Semantics* of OWL; sound calculi for the DL-based *Direct Semantics* of OWL are easily obtained by omitting these rules. Various (partial) implementations of OWL RL rule calculi have been used in distributed reasoning, discussed in more detail below. It has also been argued that rule-based reasoning is suitable for embedded devices that have very limited resources; this has been explored for both OWL EL [35] and OWL RL [93,100]. Most works on OWL RL reasoning focus on instance retrieval. Sound and complete rule-based calculi for classification in OWL RL have been developed only recently [61].

For further optimizing the application of rules, various works on OWL RL distinguish between static/pre-computed and dynamic/inferred premises of inference rules [43,106]. This can be compared to our distinction of side conditions and premises, which serves a similar purpose. The OWL RL reasoner SAOR pre-instantiates static premises (side conditions) of rules to obtain so-called *rule templates*, and indexes these templates for quick access based on the relevant dynamic premise [43]. While conceptually different, this method leads to indexing structures for rule applications similar to the ones in ELK.

In general, the efficient implementation of rule-based computations is also related to the topic of database query optimization, since rule bodies can be considered as conjunctive queries. General methods of optimizing conjunctive queries (i.e., join-project-select queries) are thus applicable; see, e.g., [1, Chapter 6]. Approaches that use a fixed set of rules like ELK can optimize join computation already when designing the algorithm, as done in Sections 5.2 and 4.3. Our concrete join implementation in ELK corresponds to a *nested loop join* that uses hash-based indexing structures to largely eliminate the inner loop. Selecting the smaller relation for the outer loop in Section 4.3 is a simple form of join order optimization.

9.3 Concurrent, Distributed, and Parallel Reasoning

Our work is not the first to address the problem of concurrent OWL reasoning. Notable earlier works include an approach for parallelizing (incomplete) structural reasoning algorithms [19], tableau procedures that explore non-deterministic choices concurrently [20, 67,70,110], a resolution calculus for $\mathcal{ALCHIQ}$ where inferences are exchanged between distributed workers [89], and a distributed classification algorithm that can be used to concurrently invoke (serial) OWL reasoners for checking relevant subsumptions [4,5]. Experimental evaluations in each case indicate potential advantages on selected examples, but further implementation and evaluation is often needed to demonstrate a clear performance advantage over state-of-the-art systems.

Several other works have studied concurrency in lightweight ontology languages. Closest to our approach is a distributed MapReduce-based algorithm for $\mathcal{EL}^+$ [77]. However, this idea has not been empirically evaluated, and it has been argued that it ignores several practical problems [90]. Saturation-based reasoning with shared memory has recently been explored for RDFS [38]. This approach also investigates the use of alternative computation platforms, such as many-core GPUs, which bears some challenges related to memory management.

Other works focus on *distributed* reasoning over many machines, instead of shared-memory parallelism on one machine. A direct approach for achieving this is to pre-partition the input and to distribute the partitions to several processing nodes for reasoning. Some form of message transfer between nodes is usually required to exchange certain inferences. Relevant theoretical results have been developed for the general case of first-order deduction [2]. Several works on partition-based ontology reasoning focus on (subsets of) OWL RL [78, 98]. Another approach to partitioning in OWL is the computation of *modules* [27], which has also been considered for distribution and related optimizations recently [3, 105].

Other prominent approaches to distributed reasoning use MapReduce as a computational framework. Many related works focus on the distribution of reasoning with assertional data using weaker schema-level modeling languages pD^* and (fragments of) RDFS [42, 59, 107, 108]. These works are distinguished from our approach by their goal to manage large-scale data (in the range of billions of axioms), which is beyond the memory capacity of a single machine. Accordingly, computation is distributed to many servers without memory sharing. Yet, we can find similarities in term-based distribution strategies [42, 43, 77, 106, 107, 108] and indexing of rules [43] with our strategy of assigning contexts to axioms.

Our abstract saturation procedure from Section 4.1 is closely related to saturation-based theorem proving [18, 109], and it may seem that concurrent extensions of this procedure as described in Section 6.2 should be known in this area. Surprisingly, this appears not to be the case. The closest to our approach is the strategy used in the theorem prover ROO [68], in which several workers apply inference and simplification rules in parallel and store the result in a shared fact database. It is assumed, however, that the access to the database is serialized, which can be the main bottleneck of the procedure when many facts are produced at the same time. More recently, from version 2.0.0, Snorocket implements a context-based concurrent procedure inspired by our approach [53].[20]

10 Conclusions

In this paper we have presented many details of the ELK reasoner ranging from theory to implementation that make ELK one of the most competitive ontology reasoning systems available today. Despite its relatively short history, ELK has already been used in many biomedical applications [37, 39, 40, 48, 81, 101], in which often it was the only reasoner that is able to handle the large volumes of data involved with a reasonable performance.

From our experiments in Section 8, we can summarize that the most significant performance improvement was due to the use of the concurrent saturation procedure (Section 6.3), achieving a speedup factor as high as 3.8. This improvement, however, may depend on the number of processors/cores available. Optimization of inference rules, in particular, avoiding decomposition of negative existential restrictions (Section 5.1) comes second. The speedup factor here was reaching 2.5. The combinations of these techniques can result in more than 8 times speedup, such as in the case of GALEN8. It is difficult to estimate the improvements gained by other optimizations, such as indexing (Section 4.2.1) or efficient joint computation (Section 4.3), and in general, by our inference rules in Fig. 3 and our approach for computing the closure using Algorithm 1, since those features cannot be easily switched off. Thus, one can only speculate about possible reasons why completion-based reasoners, such as CEL and jcel were considerably slower in our experiments even for ELK without

[20] http://protegewiki.stanford.edu/wiki/Snorocket_2.0.0

optimizations. The most likely reason seems to be the differences in strategies for rule applications, as discussed in Section 4.4. Also the difference in the inference rules may play a significant role, in particular dealing with role hierarchies as discuss in Section 3.5. Finally, the difference may be due to some poor implementation practices, such as unbounded recursive calls mentioned in Sections 4.4 and 5.5. Snorocket, for example, exhibits a much better performance even though it is claimed to implement the same procedure as CEL [66].

Many optimizations and improvements that make ELK so ‘incredible’ are not limited to just $\mathcal{EL}$. The consequence-based reasoners CB and ConDOR, which use similar techniques as ELK, support more complex (ExpTime) DLs [51,95] and yet retain a comparable performance (when disregarding concurrency). Thus, polynomial complexity of $\mathcal{EL}$ does not really explain the efficiency of our procedures in practice. In fact, the estimates obtained at the end of Section 3.5 for the rules in Fig. 3 are not even remotely similar to the numbers obtained by our experiments in Table 10. For example, for SNOMED CT the analysis would give a bound of about 90 billion unique subsumptions and 5 trillion links, whereas in our experiments even without optimizations we obtain less then 15 million and 4 million, respectively. The number of inferences predicted by this analysis is even more astronomical. If our procedure would behave according to this estimate, it would consume petabytes of memory and take millions of years to finish.

Some ideas presented in this paper are not even specific to DLs at all. For example, the abstract concurrent saturation procedure described in Section 6.2 can be used for parallel computation of the deductive closure under essentially any inference system.

This paper mainly focuses on techniques that contribute to the performance of ELK. This does not mean that there are no other interesting enhancements. For example, ELK supports interrupting and restarting of reasoning tasks, which was recently argued to be important in certain applications [35]. There is a mechanism for batch processing of saturation jobs that lets the system recognize when the saturation for an input concept is computed without waiting for all input concepts to be processed. This is used to execute other tasks in a parallel way, such as the computation of direct subsumers for concepts. While not necessarily improving performance, these features may certainly widen possible uses of ELK.

ELK is currently under heavy development. Therefore, at this time, we do not present more specific application details, such as description of the API, summary of the classes, or source code, as this information may quickly become outdated. Although this paper cannot serve as a developer manual, it can still be a good starting point for those wishing developing or using the system. ELK is an open source project, and any contribution is welcome.

There are many interesting directions for future work. Not all OWL EL features are currently covered by ELK. We have recently studied ‘pay-as-you-go’ extensions of our approach to nominals [56], but there are some technical problems yet to be solved before this feature is fully integrated into the mainstream. To support datatypes, we plan to integrate the rules for safe numerical datatypes [69]. This result can be used even with datatype restrictions outside of the OWL EL profile. The notion of context introduced in Section 6 provides a natural way to localizing inferences. This can be used not only to perform inferences in parallel, but potentially also for incremental reasoning [26], axiom pinpointing [16,83], and debugging [49,91]. Some preliminary work in this direction is already done [52].

As an interesting theoretical problem, one can mention the question of whether the notion of redundancy introduced in Section 3.2 can be generalized to other rules than $\mathsf{R}_{\exists}^{-}$. Specifically, can applications of rules R_{H} and $\mathsf{R}_{\circ}$ likewise be avoided if they produce links $E \overset{R}{\to} C$ such that $\{E \overset{R}{\to} D, D \sqsubseteq C\} \subseteq \mathsf{Closure}$ for some D?

Tractable algorithms are only a first step towards efficient ontology reasoning systems. Careful design, optimization, implementation, and analysis are at least as significant. Thus, similar to other reasoning approaches, such as tableau or resolution, implementation and optimization techniques for consequence-based procedures are important research topics. This work makes one of the first contributions to this area.

Acknowledgements This work was supported by the Royal Society, the Seventh Framework Program of the European Commission under Grant Agreement 318338, 'Optique', and the EPSRC projects ConDOR, ExODA, Score!, and MaSI[3]. The first author is supported by the German Research Council (DFG).

We also thank the International Health Terminology Standards Development Organisation (IHTSDO), and in particular Kent Spackman, for providing us with the experimental version of the ANATOMY ontology.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley (1994)
2. Amir, E., McIlraith, S.A.: Partition-based logical reasoning for first-order and propositional theories. Artificial Intelligence **162**(1-2), 49–88 (2005)
3. Armas Romero, A., Cuenca Grau, B., Horrocks, I.: MORe: modular combination of OWL reasoners for ontology classification. In: P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J.X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, E. Blomqvist (eds.) Proc. 11th Int. Semantic Web Conf. (ISWC'12), *LNCS*, vol. 7649, pp. 1–16. Springer (2012)
4. Aslani, M., Haarslev, V.: Parallel TBox classification in description logics – first experimental results. In: H. Coelho, R. Studer, M. Wooldridge (eds.) Proc. 19th European Conf. on Artificial Intelligence (ECAI'10), *Frontiers in Artificial Intelligence and Applications*, vol. 215, pp. 485–490. IOS Press (2010)
5. Aslani, M., Haarslev, V.: Concurrent classification of OWL ontologies – An empirical evaluation. In: Y. Kazakov, D. Lembo, F. Wolter (eds.) Proc. 25th Int. Workshop on Description Logics (DL'12), *CEUR Workshop Proceedings*, vol. 846, pp. 400–410. CEUR-WS.org (2012)
6. Baader, F.: Terminological cycles in a description logic with existential restrictions. In: G. Gottlob, T. Walsh (eds.) Proc. 18th Int. Joint Conf. on Artificial Intelligence (IJCAI'03), pp. 325–330. Morgan Kaufmann (2003)
7. Baader, F., Borgwardt, S., Morawska, B.: Extending unification in $\mathcal{EL}$ towards general TBoxes. In: G. Brewka, T. Eiter, S.A. McIlraith (eds.) Proc. 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'12), pp. 568–572. AAAI Press (2012)
8. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: L. Kaelbling, A. Saffiotti (eds.) Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05), pp. 364–369. Professional Book Center (2005)
9. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope further. In: K.G. Clark, P.F. Patel-Schneider (eds.) Proc. OWLED 2008 DC Workshop on OWL: Experiences and Directions, *CEUR Workshop Proceedings*, vol. 496. CEUR-WS.org (2008)
10. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications, second edn. Cambridge University Press (2007)
11. Baader, F., Franconi, E., Hollunder, B., Nebel, B., Profitlich, H.J.: An empirical analysis of optimization techniques for terminological representation systems. Applied Intelligence **4**(2), 109–132 (1994)
12. Baader, F., Küsters, R., Molitor, R.: Computing least common subsumers in description logics with existential restrictions. In: T. Dean (ed.) Proc. 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99), pp. 96–103. Morgan Kaufmann (1999)
13. Baader, F., Lutz, C., Suntisrivaraporn, B.: Is tractable reasoning in extensions of the description logic $\mathcal{EL}$ useful in practice? In: In Proceedings of the 2005 International Workshop on Methods for Modalities (M4M'05) (2005)
14. Baader, F., Lutz, C., Suntisrivaraporn, B.: CEL—a polynomial-time reasoner for life science ontologies. In: U. Furbach, N. Shankar (eds.) Proc. 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06), *LNCS*, vol. 4130, pp. 287–291. Springer (2006)
15. Baader, F., Lutz, C., Suntisrivaraporn, B.: Efficient reasoning in $\mathcal{EL}^+$. In: B. Parsia, U. Sattler, D. Toman (eds.) Proc. 2006 Int. Workshop on Description Logics (DL'06), *CEUR Workshop Proceedings*, vol. 189. CEUR-WS.org (2006)
16. Baader, F., Peñaloza, R.: Automata-based axiom pinpointing. J. of Automated Reasoning **45**(2), 91–129 (2010)
17. Baader, F., Sertkaya, B., Turhan, A.Y.: Computing the least common subsumer w.r.t. a background terminology. J. of Applied Logics **5**(3), 392–420 (2007)

18. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: J.A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, pp. 19–99. Elsevier and MIT Press (2001)
19. Bergmann, F.W., Quantz, J.: Parallelizing description logics. In: I. Wachsmuth, C.R. Rollinger, W. Brauer (eds.) Proc. 19th Annual GermanConf. on Artificial Intelligence (KI'95), *LNCS*, vol. 981, pp. 137–148. Springer (1995)
20. Bozzato, L., Homola, M., Serafini, L.: Towards more effective tableaux reasoning for CKR. In: Y. Kazakov, D. Lembo, F. Wolter (eds.) Proc. 25th Int. Workshop on Description Logics (DL'12), *CEUR Workshop Proceedings*, vol. 846, pp. 114–124. CEUR-WS.org (2012)
21. Brandt, S.: On subsumption and instance problem in $\mathcal{ELH}$ w.r.t. general TBoxes. In: V. Haarslev, R. Möller (eds.) Proc. 2004 Int. Workshop on Description Logics (DL'04), *CEUR Workshop Proceedings*, vol. 104. CEUR-WS.org (2004)
22. Brandt, S.: Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and – What else? In: R.L. de Mántaras, L. Saitta (eds.) Proc. 16th European Conf. on Artificial Intelligence (ECAI'04), pp. 298–302. IOS Press (2004)
23. Brandt, S.: Standard and non-standard reasoning in description logics. Ph.D. thesis, Technische Universität Dresden, Germany (2006)
24. Brickley, D., Guha, R.V. (eds.): RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation (10 February 2004). Available at http://www.w3.org/TR/rdf-schema/
25. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. of Automated Reasoning **39**(3), 385–429 (2007)
26. Cuenca Grau, B., Halaschek-Wiener, C., Kazakov, Y., Suntisrivaraporn, B.: Incremental classification of description logics ontologies. J. of Automated Reasoning **44**(4), 337–369 (2010)
27. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: Theory and practice. J. of Artificial Intelligence Research **31**, 273–318 (2008)
28. Day-Richter, J. (ed.): The OBO Flat File Format Specification, version 1.2. The Gene Ontology Consortium (2006). Available at http://www.geneontology.org/GO.format.obo-1_2.shtml
29. Delaitre, V., Kazakov, Y.: Classifying $\mathcal{ELH}$ ontologies in SQL databases. In: P.F. Patel-Schneider, R. Hoekstra (eds.) Proc. OWLED 2009 Workshop on OWL: Experiences and Directions, *CEUR Workshop Proceedings*, vol. 529. CEUR-WS.org (2009)
30. Dowling, W.F., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. J. of Logic Programming **1**(3), 267–284 (1984)
31. Eiter, T., Krennwallner, T., Schneider, P., Xiao, G.: Uniform evaluation of nonmonotonic DL-programs. In: T. Lukasiewicz, A. Sali (eds.) Proc. 7th Int. Symposium on Foundations of Information and Knowledge Systems (FoIKS'12), *LNCS*, vol. 7153, pp. 1–22. Springer (2012)
32. Forgy, C.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence **19**, 17–37 (1982)
33. Gkoutos, G.V., Schofield, P.N., Hoehndorf, R.: Computational tools for comparative phenomics: the role and promise of ontologies. Mammalian Genome **23**(9–10), 669–679 (2012)
34. Glimm, B., Horrocks, I., Motik, B., Shearer, R., Stoilos, G.: A novel approach to ontology classification. J. Web Sem. **14**, 84–101 (2012)
35. Grimm, S., Watzke, M., Hubauer, T., Cescolini, F.: Embedded $\mathcal{EL}^+$ reasoning on programmable logic controllers. In: P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J.X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, E. Blomqvist (eds.) Proc. 11th Int. Semantic Web Conf. (ISWC'12), *LNCS*, vol. 7649, pp. 66–81. Springer (2012)
36. Haarslev, V., Möller, R.: Racer system description. In: R. Goré, A. Leitsch, T. Nipkow (eds.) Proc. 1st Int. Joint Conf. on Automated Reasoning (IJCAR'01), *LNCS*, vol. 2083, pp. 701–705. Springer (2001)
37. Harris, M.A., Lock, A., Bühler, J., Oliver, S.G., Wood, V.: FYPO: the fission yeast phenotype ontology. Bioinformatics **29**(13), 1671–1678 (2013)
38. Heino, N., Pan, J.Z.: RDFS reasoning on massively parallel hardware. In: P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J.X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, E. Blomqvist (eds.) Proc. 11th Int. Semantic Web Conf. (ISWC'12), *LNCS*, vol. 7649, pp. 133–148. Springer (2012)
39. Hoehndorf, R., Dumontier, M., Gkoutos, G.V.: Identifying aberrant pathways through integrated analysis of knowledge in pharmacogenomics. Bioinformatics **28**(16), 2169–2175 (2012)
40. Hoehndorf, R., Harris, M.A., Herre, H., Rustici, G., Gkoutos, G.V.: Semantic integration of physiology phenotypes with an application to the cellular phenotype ontology. Bioinformatics **28**(13), 1783–1789 (2012)
41. Hofmann, M.: Proof-theoretic approach to description-logic. In: Proc. 20th Annual IEEE Symposium on Logic in Computer Science (LICS'05), pp. 229–237. IEEE Computer Society (2005)

42. Hogan, A., Harth, A., Polleres, A.: Scalable authoritative OWL reasoning for the Web. Int. J. of Semantic Web Inf. Syst. **5**(2), 49–90 (2009)
43. Hogan, A., Pan, J.Z., Polleres, A., Decker, S.: SAOR: template rule optimisations for distributed reasoning over 1 billion linked data triples. In: P.F. Patel-Schneider, Y. Pan, B. Glimm, P. Hitzler, P. Mika, J. Pan, I. Horrocks (eds.) Proc. 9th Int. Semantic Web Conf. (ISWC'10), *LNCS*, vol. 6496, pp. 337–353. Springer (2010)
44. Horridge, M., Bechhofer, S.: The OWL API: A Java API for working with OWL 2 ontologies. In: P.F. Patel-Schneider, R. Hoekstra (eds.) Proc. OWLED 2009 Workshop on OWL: Experiences and Directions, *CEUR Workshop Proceedings*, vol. 529. CEUR-WS.org (2009)
45. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: P. Doherty, J. Mylopoulos, C.A. Welty (eds.) Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06), pp. 57–67. AAAI Press (2006)
46. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for expressive description logics. In: H. Ganzinger, D.A. McAllester, A. Voronkov (eds.) Proc. 6th Int. Conf. on Logic Programming and Automated Reasoning (LPAR'99), *LNCS*, vol. 1705, pp. 161–180. Springer (1999)
47. ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. J. of Web Semantics **3**(2–3), 79–115 (2005)
48. Jupp, S., Stevens, R., Hoehndorf, R.: Logical gene ontology annotations (GOAL): exploring gene ontology annotations with OWL. J. of Biomedical Semantics **3(Suppl 1)**(S3), 1–16 (2012)
49. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: K. Aberer, K.S. Choi, N. Noy, D. Allemang, K.I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, P. Cudré-Mauroux (eds.) Proc. 6th Int. Semantic Web Conf. (ISWC'07), *LNCS*, vol. 4825, pp. 267–280. Springer (2007)
50. Kazakov, Y.: $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In: G. Brewka, J. Lang (eds.) Proc. 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'08), pp. 274–284. AAAI Press (2008)
51. Kazakov, Y.: Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In: C. Boutilier (ed.) Proc. 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09), pp. 2040–2045. IJCAI (2009)
52. Kazakov, Y., Klinov, P.: Incremental reasoning in OWL EL without bookkeeping. In: H. Alani, L. Kagal, A. Fokoue, P.T. Groth, C. Biemann, J.X. Parreira, L. Aroyo, N.F. Noy, C. Welty, K. Janowicz (eds.) Proc. 12th Int. Semantic Web Conf. (ISWC'13), *LNCS*, vol. 8218, pp. 232–247. Springer (2013)
53. Kazakov, Y., Krötzsch, M., Simančík, F.: Concurrent classification of $\mathcal{EL}$ ontologies. In: L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, E. Blomqvist (eds.) Proc. 10th Int. Semantic Web Conf. (ISWC'11), *LNCS*, vol. 7032, pp. 305–320. Springer (2011)
54. Kazakov, Y., Krötzsch, M., Simančík, F.: Unchain my $\mathcal{EL}$ reasoner. In: R. Rosati, S. Rudolph, M. Zakharyaschev (eds.) Proc. 24th Int. Workshop on Description Logics (DL'11), *CEUR Workshop Proceedings*, vol. 745, pp. 202–212. CEUR-WS.org (2011)
55. Kazakov, Y., Krötzsch, M., Simančík, F.: ELK reasoner: Architecture and evaluation. In: I. Horrocks, M. Yatskevich, E. Jimenez-Ruiz (eds.) Proc. OWL Reasoner Evaluation Workshop 2012 (ORE'12), *CEUR Workshop Proceedings*, vol. 858. CEUR-WS.org (2012)
56. Kazakov, Y., Krötzsch, M., Simančík, F.: Practical reasoning with nominals in the $\mathcal{EL}$ family of description logics. In: G. Brewka, T. Eiter, S.A. McIlraith (eds.) Proc. 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'12), pp. 264–274. AAAI Press (2012)
57. Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.A.: The Protégé OWL Plugin: An open development environment for Semantic Web applications. In: S.A. McIlraith, D. Plexousakis, F. van Harmelen (eds.) Proc. 3rd Int. Semantic Web Conf. (ISWC'04), *LNCS*, vol. 3298, pp. 229–243. Springer (2004)
58. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to ontology-based data access. In: T. Walsh (ed.) Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11), pp. 2656–2661. AAAI Press/IJCAI (2011)
59. Kotoulas, S., Oren, E., van Harmelen, F.: Mind the data skew: distributed inferencing by speeddating in elastic regions. In: Proc. 19th Int. Conf. on World Wide Web (WWW'10), WWW'10, pp. 531–540. ACM (2010)
60. Krötzsch, M.: Efficient rule-based inferencing for OWL EL. In: T. Walsh (ed.) Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11), pp. 2668–2673. AAAI Press/IJCAI (2011)
61. Krötzsch, M.: The not-so-easy task of computing class subsumptions in OWL RL. In: P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J.X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, E. Blomqvist (eds.) Proc. 11th Int. Semantic Web Conf. (ISWC'12), *LNCS*, vol. 7649, pp. 279–294. Springer (2012)
62. Krötzsch, M.: OWL 2 Profiles: An introduction to lightweight ontology languages. In: T. Eiter, T. Krennwallner (eds.) Proc. 8th Reasoning Web Summer School, Vienna, Austria, September 3–8 2012, *LNCS*, vol. 7487, pp. 112–183. Springer (2012)

63. Krötzsch, M., Mehdi, A., Rudolph, S.: Orel: Database-driven reasoning for OWL 2 profiles. In: V. Haarslev, D. Toman, G. Weddell (eds.) Proc. 23rd Int. Workshop on Description Logics (DL'10), *CEUR Workshop Proceedings*, vol. 573, pp. 114–124. CEUR-WS.org (2010)
64. Krötzsch, M., Rudolph, S., Hitzler, P.: Conjunctive queries for a tractable fragment of OWL 1.1. In: K. Aberer, K.S. Choi, N. Noy, D. Allemang, K.I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, P. Cudré-Mauroux (eds.) Proc. 6th Int. Semantic Web Conf. (ISWC'07), *LNCS*, vol. 4825, pp. 310–323. Springer (2007)
65. Krötzsch, M., Simančík, F., Horrocks, I.: A description logic primer. CoRR **abs/1201.4089** (2012)
66. Lawley, M.J., Bousquet, C.: Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In: K. Taylor, T. Meyer, M. Orgun (eds.) Proc. 6th Australasian Ontology Workshop (IAOA'10), *Conferences in Research and Practice in Information Technology*, vol. 122, pp. 45–49. Australian Computer Society Inc. (2010)
67. Liebig, T., Müller, F.: Parallelizing tableaux-based description logic reasoning. In: R. Meersman, Z. Tari, P. Herrero (eds.) Proceedings of OTM Workshops 2007, Part II, *LNCS*, vol. 4806, pp. 1135–1144. Springer (2007)
68. Lusk, E.L., McCune, W., Slaney, J.K.: Roo: A parallel theorem prover. In: D. Kapur (ed.) Proc. 11th Conf. on Automated Deduction (CADE'92), *LNCS*, vol. 607, pp. 731–734. Springer (1992)
69. Magka, D., Kazakov, Y., Horrocks, I.: Tractable extensions of the description logic $\mathcal{EL}$ with numerical datatypes. J. of Automated Reasoning **47**(4), 427–450 (2011)
70. Meissner, A.: Experimental analysis of some computation rules in a simple parallel reasoning system for the $\mathcal{ALC}$ description logic. Int. J. of Applied Mathematics and Computer Science **21**(1), 83–95 (2011)
71. Mendez, J.: jcel: A modular rule-based reasoner. In: I. Horrocks, M. Yatskevich, E. Jimenez-Ruiz (eds.) Proc. OWL Reasoner Evaluation Workshop 2012 (ORE'12), *CEUR Workshop Proceedings*, vol. 858. CEUR-WS.org (2012)
72. Mendez, J., Ecke, A., Turhan, A.Y.: Implementing completion-based inferences for the $\mathcal{EL}$-family. In: R. Rosati, S. Rudolph, M. Zakharyaschev (eds.) Proc. 24th Int. Workshop on Description Logics (DL'11), *CEUR Workshop Proceedings*, vol. 745, pp. 334–344. CEUR-WS.org (2011)
73. Mendez, J., Suntisrivaraporn, B.: Reintroducing CEL as an OWL 2 EL reasoner. In: B.C. Grau, I. Horrocks, B. Motik, U. Sattler (eds.) Proc. 22nd Int. Workshop on Description Logics (DL'09), *CEUR Workshop Proceedings*, vol. 477. CEUR-WS.org (2009)
74. Möller, R., Haarslev, V., Wandelt, S.: The revival of structural subsumption in tableau-based reasoners. In: F. Baader, C. Lutz, B. Motik (eds.) Proc. 21st Int. Workshop on Description Logics (DL'08), *CEUR Workshop Proceedings*, vol. 353. CEUR-WS.org (2008)
75. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.): OWL 2 Web Ontology Language: Profiles. W3C Recommendation (27 October 2009). Available at http://www.w3.org/TR/owl2-profiles/
76. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. of Artificial Intelligence Research **36**, 165–228 (2009)
77. Mutharaju, R., Maier, F., Hitzler, P.: A MapReduce algorithm for $\mathcal{EL}^+$. In: V. Haarslev, D. Toman, G. Weddell (eds.) Proc. 23rd Int. Workshop on Description Logics (DL'10), *CEUR Workshop Proceedings*, vol. 573, pp. 464–474. CEUR-WS.org (2010)
78. Narayanan, S., Çatalyürek, Ü.V., Kurç, T.M., Saltz, J.H.: Parallel materialization of large ABoxes. In: S.Y. Shin, S. Ossowski (eds.) Proc. ACM Symposium on Applied Computing (SAC'09), pp. 1257–1261. ACM (2009)
79. Nikitina, N., Rudolph, S.: ExpExpExplosion: uniform interpolation in general $\mathcal{EL}$ terminologies. In: L.D. Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, P.J.F. Lucas (eds.) Proc. 20th European Conf. on Artificial Intelligence (ECAI'12), *Frontiers in Artificial Intelligence and Applications*, vol. 242, pp. 618–623. IOS Press (2012)
80. Ortiz, M., Rudolph, S., Simkus, M.: Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In: F. Lin, U. Sattler, M. Truszczynski (eds.) Proc. 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'10), pp. 269–279. AAAI Press (2010)
81. Osumi-Sutherland, D., Reeve, S., Mungall, C.J., Neuhaus, F., Ruttenberg, A., Jefferis, G.S.X.E., Armstrong, J.D.: A strategy for building neuroanatomy ontologies. Bioinformatics **28**(9), 1262–1269 (2012)
82. OWL Working Group, W.: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation (27 October 2009). Available at http://www.w3.org/TR/owl2-overview/
83. Peñaloza, R., Sertkaya, B.: On the complexity of axiom pinpointing in the $\mathcal{EL}$ family of description logics. In: F. Lin, U. Sattler, M. Truszczynski (eds.) Proc. 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'10), pp. 280–289. AAAI Press (2010)
84. Rector, A., Gangemi, A., Galeazzi, E., Glowinski, A.J., Rossi-Mori, A.: The GALEN CORE model schemata for anatomy: Towards a re-usable application-independent model of medical concepts. In:

P. Barahona, M. Veloso, J. Bryant (eds.) Proc. 12th Int. Congress of the European Federation for Medical Informatics (MIE'94), pp. 229–233 (1994)
85. Rector, A., Iannone, L.: Lexically suggest, logically define: Quality assurance of the use of qualifiers and expected results of post-coordination in SNOMED CT. J. of Biomedical Informatics **45**, 199–209 (2012)
86. Rector, A.L., Bechhofer, S., Goble, C.A., Horrocks, I., Nowlan, W.A., Solomon, W.D.: The grail concept modelling language for medical terminology. Artificial Intelligence in Medicine **9**(2), 139–171 (1997)
87. Rogers, J.E.: Quality assurance of medical ontologies. Methods Inf Med. **45**(3), 267–274 (2006)
88. Rudolph, S., Krötzsch, M., Hitzler, P.: Cheap Boolean role constructors for description logics. In: S. Hölldobler, C. Lutz, H. Wansing (eds.) Proc. 11th European Conf. on Logics in Artificial Intelligence (JELIA'08), *LNAI*, vol. 5293, pp. 362–374. Springer (2008)
89. Schlicht, A., Stuckenschmidt, H.: Peer-to-peer reasoning for interlinked ontologies. Int. J. of Semantic Computing **4**(1), 27–58 (2010)
90. Schlicht, A., Stuckenschmidt, H.: MapResolve. In: S. Rudolph, C. Gutierrez (eds.) Proc. 5th Int. Conf. on Web Reasoning and Rule Systems (RR'11), *LNCS*, vol. 6902, pp. 294–299. Springer (2011)
91. Schlobach, S., Huang, Z., Cornet, R., van Harmelen, F.: Debugging incoherent terminologies. J. of Automated Reasoning **39**(3), 317–349 (2007)
92. Schulz, S., Cornet, R., Spackman, K.A.: Consolidating SNOMED CT's ontological commitment. Applied Ontology **6**(1), 1–11 (2011)
93. Seitz, C., Schönfelder, R.: Rule-based OWL reasoning for specific embedded devices. In: L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, E. Blomqvist (eds.) Proc. 10th Int. Semantic Web Conf. (ISWC'11), *LNCS*, vol. 7032, pp. 237–252. Springer (2011)
94. Sertkaya, B.: In the search of improvements to the $\mathcal{EL}^+$ classification algorithm. In: R. Rosati, S. Rudolph, M. Zakharyaschev (eds.) Proc. 24th Int. Workshop on Description Logics (DL'11), *CEUR Workshop Proceedings*, vol. 745, pp. 389–399. CEUR-WS.org (2011)
95. Simančík, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond Horn ontologies. In: T. Walsh (ed.) Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11), pp. 1093–1098. AAAI Press/IJCAI (2011)
96. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. of Web Semantics **5**(2), 51–53 (2007)
97. Smith, B., Ashburner, M., Rosse, C., Bard, J., Bug, W., Ceusters, W., Goldberg, L.J., Eilbeck, K., Ireland, A., Mungall, C.J., Consortium, T.O., Leontis, N., Rocca-Serra, P., Ruttenberg, A., Sansone, S.A., Scheuermann, R.H., Shah, N., Whetzeland, P.L., Lewis, S.: The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. Nature Biotechnology **25**, 1251–1255 (2007)
98. Soma, R., Prasanna, V.K.: Parallel inferencing for OWL knowledge bases. In: Proc. Int. Conf. on Parallel Processing (ICPP'08), pp. 75–82. IEEE Computer Society (2008)
99. Suntisrivaraporn, B.: Polynomial-time reasoning support for design and maintenance of large-scale biomedical ontologies. Ph.D. thesis, Technische Universität Dresden, Germany (2009)
100. Tai, W., Keeney, J., O'Sullivan, D.: COROR: A composable rule-entailment OWL reasoner for resource-constrained devices. In: N. Bassiliades, G. Governatori, A. Paschke (eds.) Proc. 5th Int. Conf. on Rule-Based Reasoning, Programming, and Applications (RuleML Europe'11), *LNCS*, vol. 6826, pp. 212–226. Springer (2011)
101. The Gene Ontology Consortium: Gene ontology annotations and resources. Nucleic Acids Res (2012)
102. Thomas, E., Pan, J.Z., Ren, Y.: TrOWL: Tractable OWL 2 reasoning infrastructure. In: L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, T. Tudorache (eds.) Proc. 7th Extended Semantic Web Conf. (ESWC'10), *LNCS*, vol. 6089, pp. 431–435. Springer (2010)
103. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: U. Furbach, N. Shankar (eds.) Proc. 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06), *LNCS*, vol. 4130, pp. 292–297. Springer (2006)
104. Tsarkov, D., Horrocks, I., Patel-Schneider, P.F.: Optimizing terminological reasoning for expressive description logics. J. of Automated Reasoning **39**(3), 277–316 (2007)
105. Tsarkov, D., Palmisano, I.: Chainsaw: a metareasoner for large ontologies. In: I. Horrocks, M. Yatskevich, E. Jimenez-Ruiz (eds.) Proc. OWL Reasoner Evaluation Workshop 2012 (ORE'12), *CEUR Workshop Proceedings*, vol. 858. CEUR-WS.org (2012)
106. Urbani, J., Kotoulas, S., Maassen, J., van Harmelen, F., Bal, H.: WebPIE: a Web-scale parallel inference engine using MapReduce. J. of Web Semantics pp. 59–75 (2012)
107. Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: Scalable distributed reasoning using MapReduce. In: A. Bernstein, D.R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (eds.) Proc. 8th Int. Semantic Web Conf. (ISWC'09), *LNCS*, vol. 5823, pp. 634–649. Springer (2009)

108. Weaver, J., Hendler, J.A.: Parallel materialization of the finite RDFS closure for hundreds of millions of triples. In: A. Bernstein, D.R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (eds.) Proc. 8th Int. Semantic Web Conf. (ISWC'09), *LNCS*, vol. 5823, pp. 682–697. Springer (2009)
109. Wos, L., Overbeek, R., Lusk, E., Boyle, J.: Automated Reasoning: Introduction and Applications, second edn. McGraw-Hill, Inc., New York, NY, USA (1992)
110. Wu, K., Haarslev, V.: A parallel reasoner for the description logic $\mathcal{ALC}$. In: Y. Kazakov, D. Lembo, F. Wolter (eds.) Proc. 25th Int. Workshop on Description Logics (DL'12), *CEUR Workshop Proceedings*, vol. 846, pp. 378–388. CEUR-WS.org (2012)
111. Xiang, Z., Mungall, C., Ruttenberg, A., He, Y.: Ontobee: A linked data server and browser for ontology terms. In: International Conference on Biomedical Ontologies (ICBO), pp. 279–281 (2011)
112. Xiao, G., Heymans, S., Eiter, T.: DReW: a reasoner for Datalog-rewritable description logics and dl-programs. In: T. Eiter, A.E. Ghali, S. Fernández, S. Heymans, T. Krennwallner, F. Lévy (eds.) Proc. 1st Int. Workshop on Business Models, Business Rules and Ontologies (BuRO'10), pp. 1–14. ONTORULE Project (2010)

A ABoxes and Safe Nominals

In the main parts of this paper, we have only considered terminological reasoning in $\mathcal{EL}_{\bot}^{+}$. Like other completion-based systems [21,99], our systems in Fig. 1, Fig. 2, and Fig. 3 can also be extended with rules for dealing with concept and role assertions. To avoid repeating proofs, optimizations and algorithmic details for reasoning with assertions, however, we do not present the additional rules separately. Instead, we demonstrate how ABoxes reasoning can be reduced to terminological reasoning. We also demonstrate that our reduction works for a restricted but quite commonly used pattern of nominals in OWL EL ontologies.

To this end, within this section we consider ontologies with nominals. A *nominal* is a concept of the form $C = \{a\}$ where a is an individual, which is interpreted by the singleton set $C^{\mathcal{I}} = \{a^{\mathcal{I}}\}$. We denote by $\mathcal{ELO}_{\bot}^{+}$ the extension of $\mathcal{EL}_{\bot}^{+}$ in which concepts can be constructed using nominals. Note that the assertions $C(a)$ and $R(a,b)$ are semantically equivalent to concept inclusions with nominals $\{a\} \sqsubseteq C$ and $\{a\} \sqsubseteq \exists R.\{b\}$, respectively. Within this section we also assume that the set of atomic concepts contains a distinguished atomic concept N_a for every individual a in our vocabulary. For x an $\mathcal{ELO}_{\bot}^{+}$ concept, axiom, or an ontology, we define $N(x)$ to be the result of replacing each occurrence of each nominal $\{a\}$ in x by N_a. The next lemma shows that this reduction provides us with a sufficient condition for checking entailment in $\mathcal{O}$.

Lemma 5 *Let $\mathcal{O}$ be an $\mathcal{ELO}_{\bot}^{+}$ ontology and α an $\mathcal{ELO}_{\bot}^{+}$ axiom that do not contain atomic concepts of the form N_a. Then $N(\mathcal{O}) \models N(\alpha)$ implies $\mathcal{O} \models \alpha$.*

Proof Suppose to the contrary that $N(\mathcal{O}) \models N(\alpha)$ but $\mathcal{O} \not\models \alpha$. Then there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{O}$ but $\mathcal{I} \not\models \alpha$. Let us define an interpretation $\mathcal{J}$ by setting $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$, $N_a^{\mathcal{J}} = \{a^{\mathcal{I}}\}$, $A^{\mathcal{J}} = A^{\mathcal{I}}$ for $A \neq N_a$, and $R^{\mathcal{J}} = R^{\mathcal{I}}$. Since the transformation $N(\cdot)$ merely replaces each $\{a\}$ by N_a and we have $\{a\}^{\mathcal{I}} = N_a^{\mathcal{J}}$, for every axiom β that does not contain concepts N_a we have $\mathcal{I} \models \beta$ iff $\mathcal{J} \models N(\beta)$. Since $\mathcal{I} \models \mathcal{O}$ and $\mathcal{I} \not\models \alpha$, in particular, we have $\mathcal{J} \models N(\mathcal{O})$ and $\mathcal{J} \not\models N(\alpha)$, which contradicts our assumption $N(\mathcal{O}) \models N(\alpha)$. □

As a particular case of the previous lemma, we can have the following sufficient condition for checking unsatisfiability of ontologies with assertions.

Corollary 2 *Let $\mathcal{O}$ be an $\mathcal{ELO}_{\bot}^{+}$ ontology that does not contain atomic concepts of the form N_a. If $N(\mathcal{O}) \models N_a \sqsubseteq \bot$ for some N_a, then $\mathcal{O}$ is inconsistent.*

Proof Take $\alpha := \{a\} \sqsubseteq \bot$. Clearly, α does not contain any atomic concepts of the form N_a. Since $N(\alpha) = N_a \sqsubseteq \bot$, we have $N(\mathcal{O}) \models N(\alpha)$. Therefore, by Lemma 5, $\mathcal{O} \models \alpha$. Since $\mathcal{I} \models \alpha$ for no interpretation $\mathcal{I}$, this is only possible if $\mathcal{O}$ is inconsistent. □

The converses of Corollary 2 and Lemma 5 do not hold in general, but they hold if the occurrence of nominals is restricted in the following way.

Definition 6 (Nominal Safety) An $\mathcal{ELO}_{\bot}^{+}$ concept C is *safe* if C has only occurrences of nominals in subconcepts of the form $\exists R.\{a\}$; C is *negatively safe* (short *n-safe*) if C is either safe or a nominal. A concept inclusion $C \sqsubseteq D$ is *safe* if C is n-safe and D is safe. An $\mathcal{ELO}_{\bot}^{+}$ ontology is *safe* if all its concept inclusions are safe.

The restricted use of nominals still allows to express assertion axioms $C(a)$ and $R(a,b)$ since the corresponding concept inclusions $\{a\} \sqsubseteq C$ and $\{a\} \sqsubseteq \exists R.\{b\}$ are safe. It also captures another common constructor in OWL EL ontologies ObjectHasValue(R a), which corresponds to the concept $\exists R.\{a\}$.

Theorem 4 *Let $\mathcal{O}$ be a safe $\mathcal{ELO}_\bot^+$ ontology containing no atomic concepts of the form N_a. Assume that $N(\mathcal{O}) \not\models N_a \sqsubseteq \bot$ for every N_a. Then $\mathcal{O}$ is consistent. Furthermore, for every safe concept inclusion α containing no atomic concepts of the form N_a, if $\mathcal{O} \models \alpha$, then $N(\mathcal{O}) \models N(\alpha)$.*

Proof Let $\mathsf{Closure}$ be the set of subsumptions derivable by the rules in Fig. 2 w.r.t. $N(\mathcal{O})$ and $\mathcal{J}$ the canonical model defined w.r.t. $\mathsf{Closure}$ according to Definition 2. Since the rules in Fig. 2 are sound, we have $N_a \sqsubseteq \bot \notin \mathsf{Closure}$ for every N_a, thus, $\mathcal{J}$ is well-defined, and its domain contains a distinguished element $x_{N_a} \in N_a^{\mathcal{J}}$ for every individual a. Since $\mathsf{Closure}$ is closed under the rules in Fig. 2 (and thus closed under $\mathsf{R}_\exists^-$ up to redundancy), by Theorem 1, $\mathcal{J} \models N(\mathcal{O})$. Define an interpretation $\mathcal{I}$ with $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$, $A^{\mathcal{I}} = A^{\mathcal{J}}$ for atomic concepts, $R^{\mathcal{I}} = R^{\mathcal{J}}$ for roles, and $a^{\mathcal{I}} = x_{N_a}$ for the individuals. Then for every $\mathcal{ELO}_\bot^+$ concept D we have:

Claim (i) if D is safe then $D^{\mathcal{I}} = N(D)^{\mathcal{J}}$.

Claim (ii) if D is n-safe then $D^{\mathcal{I}} \subseteq N(D)^{\mathcal{J}}$.

The proof of Claim (i) is by induction on the structure of D. The only non-trivial case in the induction is for $D = \exists R.\{a\}$. To prove that $(\exists R.\{a\})^{\mathcal{I}} \subseteq (\exists R.N_a)^{\mathcal{J}}$, consider any $x_C \in (\exists R.\{a\})^{\mathcal{I}}$. Since $a^{\mathcal{I}} = x_{N_a}$, we have $\langle x_C, x_{N_a}\rangle \in R^{\mathcal{I}} = R^{\mathcal{J}}$. Since $x_{N_a} \in N_a^{\mathcal{J}}$ by Corollary 1, the desired $x_C \in (\exists R.N_a)^{\mathcal{J}}$ follows. To prove that $(\exists R.N_a)^{\mathcal{J}} \subseteq (\exists R.\{a\})^{\mathcal{I}}$, consider any $x_C \in (\exists R.N_a)^{\mathcal{J}}$. Then there exists $x_E \in \Delta^{\mathcal{J}}$ such that $\langle x_C, x_E\rangle \in R^{\mathcal{J}}$ and $x_E \in N_a^{\mathcal{J}}$. Then by the definition of the canonical model, $C \overset{R}{\to} E \in \mathsf{Closure}$ and $E \sqsubseteq N_a \in \mathsf{Closure}$. Due to closure under $\mathsf{R}_\exists^+$ and $\mathsf{R}_\exists^-$, $C \overset{R}{\to} N_a \in \mathsf{Closure}$, and so $\langle x_C, x_{N_a}\rangle \in R^{\mathcal{J}}$ as well. Since $a^{\mathcal{I}} = x_{N_a}$ and $R^{\mathcal{I}} = R^{\mathcal{J}}$, the desired $x_C \in (\exists R.\{a\})^{\mathcal{I}}$ follows. This concludes the proof of (i).

Claim (ii) follows immediately from (i) if D is safe. Otherwise, $D = \{a\}$ is a nominal and $N(D) = N_a$; in this case we have $D^{\mathcal{I}} = \{a^{\mathcal{I}}\} = \{x_{N_a}\} \subseteq N_a^{\mathcal{J}} = N(D)^{\mathcal{J}}$, as required.

It is now easy to show that $\mathcal{I} \models \mathcal{O}$. Indeed, for every role inclusion or role composition axiom $\alpha \in \mathcal{O}$, since $\mathcal{J} \models N(\mathcal{O})$, $N(\alpha) = \alpha$ and $\mathcal{I}$ interprets roles like $\mathcal{J}$, we have $\mathcal{I} \models \alpha$. It remains to show that $\mathcal{I} \models \alpha$ for every concept inclusion $\alpha = C \sqsubseteq D \in \mathcal{O}$. By the assumption of the theorem each such α is safe, i.e., C is n-safe and D is safe. Then by Claim (ii), $C^{\mathcal{I}} \subseteq N(C)^{\mathcal{J}}$ and by Claim (i), $D^{\mathcal{I}} = N(D)^{\mathcal{J}}$. Since $\mathcal{J} \models N(\mathcal{O})$ and $\alpha \in \mathcal{O}$, we have $N(C)^{\mathcal{J}} \subseteq N(D)^{\mathcal{J}}$. Therefore, $C^{\mathcal{I}} \subseteq N(C)^{\mathcal{J}} \subseteq N(D)^{\mathcal{J}} = D^{\mathcal{I}}$, hence $\mathcal{I} \models C \sqsubseteq D$. This proves that $\mathcal{I}$ is a model of $\mathcal{O}$ and, in particular, $\mathcal{O}$ is consistent.

To conclude the proof of the theorem, let $\alpha = C \sqsubseteq D$ be an arbitrary safe concept inclusion such that $\mathcal{O} \models \alpha$. If $N(C) \sqsubseteq \bot \in \mathsf{Closure}$ then clearly $N(\mathcal{O}) \models N(\alpha)$ because the inference rules in Fig. 2 are sound. Otherwise, $x_{N(C)} \in \Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$. We show that $x_{N(C)} \in C^{\mathcal{I}}$. Indeed, if C is safe then $x_{N(C)} \in N(C)^{\mathcal{J}} = C^{\mathcal{I}}$ by Corollary 1 and Claim (i); otherwise, if $C = \{a\}$, we have $x_{N(C)} = x_{N_a} = a^{\mathcal{I}} \in \{a^{\mathcal{I}}\} = C^{\mathcal{I}}$. Now, since $\mathcal{I} \models \mathcal{O} \models \alpha$, we have $x_{N(C)} \in C^{\mathcal{I}} \subseteq D^{\mathcal{I}} = N(D)^{\mathcal{J}}$. Hence, by Lemma 2, we have $N(C) \sqsubseteq N(D) \in \mathsf{Closure}$, and thus $N(\mathcal{O}) \models N(C) \sqsubseteq N(D)$. □

Remark 2 Theorem 4 fails if the use of nominals is not safe. Take, for example, the ontology

$$\mathcal{O} = \{A \sqsubseteq \{a\}, B \sqsubseteq \{a\}, A \sqsubseteq \exists R.B\}.$$

Clearly, $\mathcal{O}$ is consistent. Moreover, $\mathcal{O} \models A \sqsubseteq B$ since for every model $\mathcal{I}$ of $\mathcal{O}$ we have either $A^{\mathcal{I}} = \emptyset$ or $A^{\mathcal{I}} = B^{\mathcal{I}} = \{a^{\mathcal{I}}\}$, and so, $\mathcal{I} \models A \sqsubseteq B$. However, for $N(\mathcal{O}) = \{A \sqsubseteq N_a, B \sqsubseteq N_a, A \sqsubseteq \exists R.B\}$, we have $N(\mathcal{O}) \not\models A \sqsubseteq B = N(A \sqsubseteq B)$.

Remark 3 Note that if N_a does not occur in $N(\mathcal{O})$, then $N(\mathcal{O}) \models N_a \sqsubseteq \bot$ iff $N(\mathcal{O}) \models \top \sqsubseteq \bot$. Therefore, in Theorem 4 it is sufficient to test whether $N(\mathcal{O}) \not\models N_a \sqsubseteq \bot$ only for the individuals a occurring in $\mathcal{O}$ or, if $\mathcal{O}$ contains no individuals, whether $N(\mathcal{O}) = \mathcal{O} \not\models \top \sqsubseteq \bot$.

By combining Theorems 3 and 4, we can describe a 'one pass' ontology realization procedure for computing all entailed instances of atomic concepts occurring in a safe $\mathcal{ELO}_\bot^+$ ontology $\mathcal{O}$. This can be accomplished by computing the closure of Input containing $\mathsf{init}(N_a)$ for every individual a occurring in $\mathcal{O}$ under the rules in Fig. 3 w.r.t. $N(\mathcal{O})$. If $N_a \sqsubseteq \bot$ is derived for at least one individual a, then $\mathcal{O}$ is inconsistent. Otherwise, for every atomic concept A, the derived subsumptions of the form $N_a \sqsubseteq A$ correspond exactly to the entailed instances $A(a)$. As in the case of ontology classification, this procedure can be implemented in polynomial time.