Consequence-Based Reasoning for Ontology Classification

František Simančík
Worcester College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity 2013

Abstract

Description logics (DLs) are knowledge representation languages that provide the theoretical underpinning for modern ontology languages such as OWL and serve as the basis for the development of ontology reasoners and tools.

Most modern ontology reasoners are based on optimized tableau algorithms, which perform reasoning by trying to build counter-models. More recently, another kind of reasoning algorithms has been introduced that, instead of building counter-models, directly derive logical consequences of axioms in the ontology using inference rules. Such consequence-based algorithms were first introduced for the $\mathcal{EL}$ family of DLs, and later extended to more expressive Horn DLs. However, up until now, consequence-based algorithms could not handle non-Horn features such as disjunctions.

We consider several complementary aspects of consequence-based reasoning in this thesis. Firstly, we describe the parallelized consequence-based reasoner ELK, which is currently the fastest reasoner for $\mathcal{EL}$ ontologies. Secondly, we demonstrate how consequence-based algorithms can be extended to handle disjunctions using inference rules reminiscent of ordered resolution. Finally, we combine our consequence-based framework with methods based on tree decompositions, and thus obtain what we believe are the first fixed-parameter tractability results for subsumption reasoning in DLs.

Acknowledgements

First of all, this thesis would not have been possible without the help of my supervisors Ian Horrocks and Yevgeny Kazakov. I am grateful to Ian for always being able to find time for me when I needed advice, both technical and about academia in general, and for checking the drafts of this thesis. I thank Yevgeny for introducing me to consequence-based reasoning and having patience with me when I struggled with our first papers and reasoners.

I am further grateful to Boris Motik and Markus Krötzsch whose ideas and contributions substantially influenced this thesis. Their diligent work has been a great source of motivation for me.

I also thank Birte Glimm who, together with Yevgeny, took excellent care of me during my stay in Ulm. Thanks to them, I retain many warm memories from that trip.

Finally, I thank the members of the Department of Computer Science, especially the KRR group and the Slovak Mafia, for many interesting discussions over lunch and coffee breaks. I had great time with all of you.

This work was supported by a studentship under the Engineering and Physical Sciences Research Council (EPSRC) Doctoral Training Award scheme.

Contents

Part I

Foundations

A beginning is the time for taking the most delicate care that the balances are correct.

—Frank Herbert
Dune

Chapter 1

Introduction

Description logics (DLs) [12] are a family of formal knowledge representation languages that provide the theoretical underpinning for modern ontology languages such as OWL [29] and serve as the basis for the development of ontology reasoning algorithms and tools. DLs have been employed extensively in areas as diverse as biology [99, 117, 122], medicine [40, 113], geography [42], astronomy [33], agriculture [123], and defence [77].

To deal effectively with a large number of concepts involved in modern ontologies, the concepts are typically organized in a hierarchical structure called a *taxonomy* which reflects the *subsumption relation* between the concepts. In a medical ontology, for example, 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 reduce redundancies in modeling of taxonomies: rather than stating all relations between the concepts explicitly, an ontology engineer provides definitions of concepts and their general properties, from which the subsumption relations can be computed using an appropriate reasoning algorithm [106]. Reasoning also plays an important role during the design of ontologies (e.g., for detecting inconsistencies and other modeling errors [103]) and in the deployment of ontologies (e.g., for query answering [99]). An important terminological reasoning task in DLs is *ontology classification* whose goal is to compute the taxonomy.

Most modern ontology reasoners, such as FaCT++ [132], HermiT [94], Pellet [121], and RacerPro [47], implement optimized *tableau-based* algorithms, or variations thereof, which perform classification by trying to build counter-models for candidate subsumptions. More recently, another kind of reasoning algorithms has been introduced which, instead of building counter-models, di-

rectly derive logical consequences of axioms in the ontology using inference rules. For this reason, such algorithms are sometimes called *consequence-based*. The inference rules are designed to derive all implied subsumptions, while guaranteeing that only a bounded number of consequences is derived.

1.1 Consequence-Based Reasoning

Consequence-based classification algorithms were first introduced for the tractable $\mathcal{EL}$ family of DLs [10]. $\mathcal{EL}$ is a simple DL which features top ($\top$), conjunctions ($C \sqcap D$), and existential restrictions ($\exists R.C$) as the only concept constructors [8].[1] These, however, are amongst the most common constructors used in existing ontologies. Some of the largest currently available ontologies, such as SNOMED CT [113]—a medical ontology describing over 300,000 concepts—can be expressed in a minor extension of $\mathcal{EL}$. Other commonly used ontologies, such as Open Biomedical Ontologies (OBO) [122], are covered by $\mathcal{EL}$ to a large degree.

The consequence-based classification algorithm for $\mathcal{EL}$ has many characteristics that can result in much better practical performance than conventional tableau-based algorithms. Since the consequence-based algorithm derives only consequences of the ontology, it never checks subsumptions that are not entailed. The number of entailed subsumptions is typically much smaller than the number of all potential subsumptions between the concepts. For example, SNOMED CT entails only about 5 million subsumptions, which is less than 0.01% of the total number of possible subsumptions. Furthermore, the algorithm derives all entailed subsumptions in 'one pass', and the number of operations needed for classification is much smaller than for computing each entailed subsumption separately. Finally, as we will show, the algorithm is relatively easily parallelizable.

Although modern tableau-based reasoners incorporate many optimizations that can reduce the number of subsumption tests and reuse the results of computations between the tests [39, 89, 133], they still cannot achieve the performance of consequence-based reasoners on $\mathcal{EL}$ ontologies. For example, the $\mathcal{EL}$ version of GALEN [102]—an average size ontology containing about 23,000 concepts—cannot be classified by any tableau reasoner available today, but can easily be classified by all existing $\mathcal{EL}$ reasoners. The main difficulty for tableau reasoners is that GALEN contains

[1] DL constructors will be introduced formally in Chapter 2.

many cyclic axioms, which causes them to construct very large models.

It has recently been shown that consequence-based classification algorithms are not limited to just $\mathcal{EL}$, but can be extended to more expressive Horn DLs [60, 98]. In addition to the favorable properties above, the extended algorithms demonstrate optimal worst-case complexity and 'pay-as-you-go' behavior: the fewer non-$\mathcal{EL}$ constructors an ontology uses, the more the algorithm behaves like the $\mathcal{EL}$ algorithm. However, up until now, consequence-based algorithms could not handle non-Horn features such as disjunctions.

1.2 Contributions

We consider several complementary aspects of consequence-based reasoning in this thesis. Firstly, in Part II, we describe the parallelized consequence-based reasoner ELK, which is currently the fastest reasoner for $\mathcal{EL}$ ontologies. Secondly, in Part III, we demonstrate how consequence-based algorithms can be extended to handle disjunctions using inference rules reminiscent of ordered resolution. Finally, in Part IV, we combine our consequence-based framework with methods based on tree decompositions, and thus obtain what we believe are the first fixed-parameter tractability results for subsumption reasoning in DLs.

1.2.1 Part II: ELK System Description

In this part of the thesis we present ELK—an open source Java-based reasoner for OWL EL ontologies.[2] OWL EL is a profile of the W3C standardized logic-based ontology language OWL [90, 100] based on the $\mathcal{EL}$ family of DLs. The main goals of ELK are good 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 [99], to integrate databases of diseases, genes, and drugs [50, 51], and to validate and query genetic ontologies [58, 129]. The latest stable release 0.3.1 of ELK supports a fragment of OWL EL that corresponds to the DL $\mathcal{EL}^+_\bot$ which extends $\mathcal{EL}$ with bottom ($\bot$) and role inclusion axioms.

Although the algorithm in ELK shares many similarities with existing $\mathcal{EL}$ algorithms [10], it offers a range of improvements. First, the algorithm does not require the input ontology to contain

[2] `http://elk.semanticweb.org/`

only simple (flattened) axioms. Second, the algorithm can avoid many redundant inferences. Third, the algorithm 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 [15] and jcel [87], and in about 40 seconds by the $\mathcal{EL}$ reasoner Snorocket [78]. The same ontology can now be classified by ELK in as little as 5 seconds on the same (quad-core) computer.

- In Chapter 3 we present a consequence-based calculus for terminological reasoning with $\mathcal{EL}^+_\bot$ ontologies, and prove its soundness and completeness. The calculus is closely related to existing completion-based algorithms for the $\mathcal{EL}$ family [10, 11, 25], but it can be used without normalizing the ontology.

- In Chapter 4 we describe a multi-phase algorithm that implements this calculus using indexing and an abstract saturation procedure. We then present a concurrent extension of this procedure, in which several independent workers can apply inference rules in parallel. Notably, our concurrent procedure requires neither locking nor thread-safe datastructures for storing conclusions.

- In Chapter 5 we describe some key optimizations implemented in ELK, in particular, specific join optimizations for matching premises of the rules, optimized transitive reduction, and special treatment of disjointness axioms. We also demonstrate that some applications of inference rules in our algorithm can be avoided without losing completeness, introduce and study an appropriate notion of redundancy for inferences, and discuss how it can be applied in practice.

- In Chapter 6 we provide an extensive experimental evaluation measuring the effect of concurrency and 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. We also compare the performance of ELK with

other commonly used DL reasoners.

- In Chapter 7 we discuss related works and conclude with suggestions for future work.

1.2.2 Part III: Beyond Horn DLs

Consequence-based classification algorithms are not limited to just $\mathcal{EL}$, or even to tractable DLs, but can be extended to more expressive DLs, such as Horn-$\mathcal{SHIQ}$ [60] and even Horn-$\mathcal{SROIQ}$ [98], which approximately corresponds to the Horn fragment of OWL. However, up until now, consequence-based algorithms could not handle non-Horn features such as disjunctions. In tableau algorithms, disjunctive axioms such as $A \sqsubseteq B \sqcup C$ result in non-deterministic inferences: in order to satisfy A one tries to satisfy either B or C. A direct reformulation of this idea as a non-deterministic rule producing consequences would not work: if $A \sqsubseteq B \sqcup C$ holds then it is not true that either $A \sqsubseteq B$ or $A \sqsubseteq C$ holds.

In this part of the thesis we demonstrate how disjunctions can be handled in a deterministic way using inference rules reminiscent of ordered resolution (see, e.g., [18]). We will consider a relatively simple DL $\mathcal{ALCI}$ which extends $\mathcal{EL}$ with disjunctions ($C \sqcup D$), negations ($\neg C$), universal restrictions ($\forall R.C$), and inverse roles (R^-). Moreover, via well-known reductions of transitivity and role inclusion axioms, our algorithm can also be applied to $\mathcal{SHI}$ ontologies, thus covering a significant subset of OWL. Our extended algorithm retains many favorable properties of existing consequence-based algorithms, including optimal worst-case complexity, one-pass classification, and pay-as-you-go behavior.

Disjunctions are interesting not only from a theoretical point of view. Although many existing ontologies are Horn, in particular the largest ones such as SNOMED CT and GALEN, this is often for historical reasons, and advances in reasoning systems for expressive DLs have led many ontology developers to consider the use of new language features. One example of this phenomenon is the latest initiative to remodel the anatomical part of SNOMED CT. Presently, the ontology uses the so-called SEP-triplet encoding [125], which encodes 'part-of' relations as 'is-a' relations. For example, 'finger' is modeled using a triple of concepts: S-finger representing the *structure* of finger, which subsumes E-finger representing the *entire* finger and P-finger representing the *parts* of finger. The fact that finger is a part of hand is expressed as S-finger $\sqsubseteq$ P-hand. A new version of

the SNOMED CT anatomical model is being developed using axioms that fully define the S- and P- concepts using disjunctions and the transitive part-of relation, for example:

$$\mathsf{S\text{-}finger} \equiv \mathsf{E\text{-}finger} \sqcup \mathsf{P\text{-}finger},$$
$$\mathsf{P\text{-}finger} \equiv \exists \mathsf{part\text{-}of}.\mathsf{E\text{-}finger}.$$

We have been granted access to a preliminary version of the ontology featuring this encoding, and were able to classify it in under 2 minutes using our new algorithm. In comparison, the fastest tableau-based reasoner required over 25 minutes to classify this ontology.

- In Chapter 8 we present a consequence-based algorithm for subsumption reasoning in $\mathcal{ALCI}$ and $\mathcal{SHI}$. We prove its soundness and completeness, describe optimizations, and present first experimental results which suggest that the algorithm scales well to non-Horn ontologies without adversely affecting performance on Horn ontologies.

1.2.3 Part IV: Parameterized Reasoning in DLs

Subsumption checking in $\mathcal{ALCI}$ is ExpTime-complete [131]. The reason for this high complexity is the size and the number of candidate models [12, Ch. 3]: due to an interaction between existential and universal quantifiers, an ontology may be satisfied only in models containing an exponential number of objects (an effect known as *and-branching*), and due to disjunctions, one may need to consider an exponential number of candidate models (an effect known as *or-branching*). State of the art DL reasoners are heavily optimized to curb and- and or-branching [12, Ch. 9], which allows them to successfully process many large and nontrivial ontologies. The performance of such reasoners is, however, relatively brittle. Although there has been some work on identifying the *qualitative* features of an ontology that may degrade the performance of a given reasoner [41], no existing method can provide a *quantitative* measure of the difficulty of reasoning with a particular ontology.

One can frame this problem in terms of *parameterized complexity* [34], which measures the difficulty of a computational problem not only w.r.t. the *size n* of a problem instance, but also a *parameter k* that quantifies specific aspects of the instance. Of particular interest are *fixed-parameter tractable* (FPT) problems, which can be solved in time $f(k) \cdot n^c$ for a fixed constant c and a fixed

computable function f; such problems are interesting because one can hope to solve large instances provided that the parameter remains small. For example, many hard graph-theoretic problems (including graph homomorphism and 3-colorability) are FPT when the parameter is the graph's *treewidth* [105]—a measure of the graph's similarity to a tree. Furthermore, the notion of treewidth has also been extended to propositional formulae and has been used to obtain FPT results [126].

In this part of the thesis we combine our consequence-based algorithm from Part III with methods based on tree decompositions [105], and thus obtain what we believe to be the first framework for a quantitative, parameterized analysis of the complexity of subsumption reasoning in DLs. We characterize the difficulty of subsumption reasoning with an $\mathcal{ALCI}$ ontology using a graph-like structure called a *decomposition*, and present a family of reasoning algorithms based on decompositions that facilitate FPT reasoning.

- In Chapter 9 we introduce the central notion of a *decomposition* $\mathcal{D}$ of an ontology $\mathcal{O}$ and a set of queries $\mathcal{Q}$. Intuitively, $\mathcal{D}$ is a graph-like structure that summarizes the models of $\mathcal{O}$ relevant for answering the queries in $\mathcal{Q}$; each vertex of $\mathcal{D}$ identifies a propositional subproblem that could occur in these models, and each edge of $\mathcal{D}$ identifies a pathway for the exchange of information between such subproblems. We identify the *length* and *width* of $\mathcal{D}$ as parameters that characterize the and- and or-branching encountered during the application of our consequence-based algorithm from Part III to $\mathcal{O}$ and $\mathcal{Q}$. Finally, we show that the algorithm can be implemented so that it is FPT w.r.t. decomposition length and width.

- In Chapter 10 we further extend our results using methods based on tree decompositions. More specifically, we introduce a notion of an *ϵ-refinement* of a decomposition $\mathcal{D}$ which, roughly speaking, is obtained by replacing each vertex of $\mathcal{D}$ with a tree decomposition of the propositional problem corresponding to the vertex. This can reduce the width of a decomposition while increasing the length by a linear factor, so it can reduce the complexity of reasoning. Computing an ϵ-refinement of $\mathcal{D}$ of bounded width (if one exists) is FPT.

- In Chapter 11 we present transformations for decomposition optimization, and establish lower and upper bounds on the sizes of the decompositions of $\mathcal{O}$ and $\mathcal{Q}$. For the lower bound, we show that $\mathcal{O}$ and $\mathcal{Q}$ exist for which all decompositions of minimal width have exponential

length. For the upper bound, we show that $\mathcal{O}$ and $\mathcal{Q}$ always admit a decomposition of minimal width with at most exponential length.

- In Chapter 12 we measure decomposition width and length of realistic ontologies. Our results show that almost all ontologies admit a decomposition of length about twice the number of concepts in the ontology and of width below 30. These results, we believe, explain the good performance of consequence-based reasoning algorithms in practice. We conclude with suggestions for future work.

Deep in the human unconscious is a pervasive need for a logical universe that makes sense. But real universe is always one step beyond logic.

—Frank Herbert
Dune

Chapter 2

Description Logics

Description logics (DLs) [12] are a family of knowledge representation languages with formal syntax and semantics that are widely used in ontological modeling. An important reason for this is that they provide one of the main underpinnings for the Web Ontology Language OWL [29]. However, DLs have been used in knowledge representation long before the advent of ontological modeling in the context of the Semantic Web, with first DL modeling languages arising in the mid 1980s [24, 111] as an attempt to standardize the semantics of network-based knowledge representation systems [79]. Many DLs can be seen as decidable fragments of first-order logic [3] and as syntactic variants of modal logics [108].

DLs provide means to model the relationships between entities in a domain of interest. In DLs there are three kinds of entities: concepts, roles, and named individuals. Concepts denote sets of individuals, roles denote binary relations between the individuals, and named individuals denote single individuals in the domain. A DL ontology consists of a set of statements, called axioms, that are known to be true in the domain. These axioms typically capture only partial knowledge about the domain that the ontology is describing, and there may be many different states of the world that are consistent with the ontology. It is customary to separate axioms into two groups. Assertional (ABox) axioms capture factual knowledge about named individuals, such as membership of an individual in a concept or a relationship between two individuals via a role. Terminological (TBox) axioms describe general knowledge about relationships between concepts and roles.

For example, an ontology modeling the domain of people and their family relationships might use concepts such as Woman to represent the set of all women, roles such as parentOf to denote

the (binary) relationship between parents and their children, and individual names such as julia and john to denote the individuals Julia and John. The ontology might include ABox axioms such as Woman(julia) and parentOf(julia, john) to assert that Julia is a woman and that Julia is a parent of John, and TBox axioms such as Mother ≡ Woman ⊓ ∃parentOf.Person to state that mothers are exactly those individuals who are women and are parents of at least one person.

DLs are equipped with a formal semantics that gives a precise specification of the meaning of DL ontologies. This semantics allows humans and computer systems to exchange ontologies without ambiguity as to their intended meaning, and also makes it possible to use logical deduction to infer implicit information from the facts stated explicitly in an ontology. The computation of inferences is called reasoning and an important goal of DL language design has been to ensure that reasoning algorithms of good performance are available. This is one of the reasons why there is not just a single description logic: the best balance between expressivity of the language and complexity of reasoning depends on the intended application.

In this chapter we introduce some of the DLs that are most commonly used in practice and describe their relationship to the Web Ontology Language OWL. Finally, we list and discuss the DLs that are used later in this thesis.

2.1 Syntax and Semantics

Here we formally introduce syntax and semantics of description logics (DLs). Many different DL features have been introduced in the literature; we present those that correspond to the DL $\mathcal{SROIQ}$ [56], which is the logic that underlies OWL. Readers who are not familiar with DLs may wish to consult a more gentle first introduction [76].

DLs are defined w.r.t. to a fixed vocabulary Σ consisting of countably infinite sets Σ_I of *named individuals*, Σ_T of *atomic roles*, and Σ_A of *atomic concepts*. Complex *roles* and *concepts* are defined recursively in Table 2.1. Unless specified otherwise, we use the letters a, b for individuals, R, S for roles, T for atomic roles, C, D, E for concepts, and A, B for atomic concepts. At-least restrictions $\geqslant n\, R.C$ and at-most restrictions $\leqslant n\, R.C$ are jointly called *number restrictions*. For convenience, for an arbitrary role R we define its *inverse* $\mathsf{inv}(R)$ by $\mathsf{inv}(T) := T^-$ and $\mathsf{inv}(T^-) := T$.

DL axioms are listed in Table 2.2. Additionally, a *concept equivalence* $C \equiv D$ abbreviates

	syntax	semantics
Individuals:		
named individual	a	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Roles:		
atomic role	T	$T^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
inverse role	T^-	$\{\langle x, y\rangle \mid \langle y, x\rangle \in T^{\mathcal{I}}\}$
Concepts:		
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
existential restriction	$\exists R.C$	$\{x \mid$ some $R^{\mathcal{I}}$-successor of x is in $C^{\mathcal{I}}\}$
universal restriction	$\forall R.C$	$\{x \mid$ all $R^{\mathcal{I}}$-successors of x are in $C^{\mathcal{I}}\}$
at-least restriction	$\geqslant n\, R.C$	$\{x \mid$ at least n $R^{\mathcal{I}}$-successors of x are in $C^{\mathcal{I}}\}$
at-most restriction	$\leqslant n\, R.C$	$\{x \mid$ at most n $R^{\mathcal{I}}$-successors of x are in $C^{\mathcal{I}}\}$
local reflexivity	$\exists R.Self$	$\{x \mid \langle x, x\rangle \in R^{\mathcal{I}}\}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$

Table 2.1: Syntax and semantics of DL constructors

the two concept inclusions $C \sqsubseteq D$ and $D \sqsubseteq C$, and a *role equivalence* $R \equiv S$ is used similarly. Axioms that involve individuals are called *assertional axioms*, or *ABox axioms*. Other axioms are called *terminological axioms*, or *TBox axioms*. Role inclusions and compositions are sometimes jointly called *complex role inclusions*. The axioms in the bottom part of Table 2.2 are called *role characteristics*. An ontology $\mathcal{O}$ is a finite set of axioms. The *size* of $\mathcal{O}$, written $\|\mathcal{O}\|$, is the number of symbols in $\mathcal{O}$. The *cardinality* of $\mathcal{O}$, written $|\mathcal{O}|$, is the number of axioms in $\mathcal{O}$.

DLs have 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 named individual a an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, to each atomic role T a binary relation $T^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to each atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$. This assignment is extended to complex roles and concepts as shown in Table 2.1, where by an "$R^{\mathcal{I}}$-successor of x" we mean any domain element y such that $\langle x, y\rangle \in R^{\mathcal{I}}$.

An interpretation $\mathcal{I}$ *satisfies* an axiom α (written $\mathcal{I} \models \alpha$) if the corresponding condition in Table 2.2 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.

	syntax	semantics
ABox:		
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$R(a,b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$
individual equality	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
individual inequality	$a \not\approx b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$
TBox:		
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	$R_1 \circ R_2 \sqsubseteq S$	$\langle x,y \rangle \in R_1^{\mathcal{I}} \wedge \langle y,z \rangle \in R_2^{\mathcal{I}} \rightarrow \langle x,z \rangle \in S^{\mathcal{I}}$
transitive role	$\mathsf{Tra}(R)$	$\langle x,y \rangle \in R^{\mathcal{I}} \wedge \langle y,z \rangle \in R^{\mathcal{I}} \rightarrow \langle x,z \rangle \in R^{\mathcal{I}}$
symmetric role	$\mathsf{Sym}(R)$	$\langle x,y \rangle \in R^{\mathcal{I}} \rightarrow \langle y,x \rangle \in R^{\mathcal{I}}$
asymmetric role	$\mathsf{Asy}(R)$	$\langle x,y \rangle \in R^{\mathcal{I}} \rightarrow \langle y,x \rangle \notin R^{\mathcal{I}}$
reflexive role	$\mathsf{Ref}(R)$	$\langle x,x \rangle \in R^{\mathcal{I}}$ for every $x \in \Delta^{\mathcal{I}}$
irreflexive role	$\mathsf{Irr}(R)$	$\langle x,x \rangle \notin R^{\mathcal{I}}$ for every $x \in \Delta^{\mathcal{I}}$
universal role	$\mathsf{Uni}(R)$	$\langle x,y \rangle \in R^{\mathcal{I}}$ for all $x,y \in \Delta^{\mathcal{I}}$
disjoint roles	$\mathsf{Dis}(R,S)$	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$

Table 2.2: Syntax and semantics of DL axioms

$\mathcal{O}$ 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 DLs 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 case α is a concept inclusion ($\alpha = C \sqsubseteq D$), the problem is known as *subsumption checking*. Many other reasoning problems can be reduced to subsumption checking. In any DL that supports $\top$ and $\bot$, consistency checking ($\alpha = \top \sqsubseteq \bot$) and satisfiability checking ($\alpha = C \sqsubseteq \bot$) are just special cases of subsumption checking. In any DL that supports existential restrictions, entailment of a role inclusion $R \sqsubseteq S$ can be checked by checking the subsumption $\exists R.A \sqsubseteq \exists S.A$ where A is a new atomic concept not occurring in the ontology (see, e.g., [39]).

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*: an acyclic graph representing direct subsumptions between equivalence classes of atomic concepts occurring in $\mathcal{O}$. The goal of the *ontology realization* task is to compute all entailed instances of atomic concepts occurring in $\mathcal{O}$.

2.2 The DL Family

Different DLs can typically be characterized by the types of constructors and axioms they allow. For example, the description logic $\mathcal{ALC}$ [111] is the fragment of $\mathcal{SROIQ}$ that only allows the constructors $\top$, $\bot$, $\sqcap$, $\sqcup$, $\neg$, $\exists$, $\forall$, and only concept inclusion axioms. The extension of $\mathcal{ALC}$ with transitive roles is traditionally denoted by the letter $\mathcal{S}$ [57]. Additional letters in DL names hint at a particular constructor or an axiom, such as inverse roles $\mathcal{I}$, number restrictions $\mathcal{Q}$, nominals $\mathcal{O}$, and role inclusions $\mathcal{H}$. For example, the DL $\mathcal{SHI}$ extends $\mathcal{S}$ with role inclusions and inverse roles. The letter $\mathcal{R}$ is most commonly used for the combination of complex role inclusions, role characteristics, and local reflexivity. This naming scheme explains the name of $\mathcal{SROIQ}$. Furthermore, each logic can be extended with a number of *concrete domains* (a.k.a. datatypes) such as Booleans, integers, and strings. This is commonly denoted by the letter $\mathcal{D}$ in parentheses as in $\mathcal{SROIQ}(\mathcal{D})$. Each logic can also be considered with or without ABox axioms.

2.2.1 Syntactic Restrictions

Unrestricted combination of the above DL constructors makes reasoning undecidable. To recover decidability, expressive DLs impose additional syntactic restrictions on ontologies. These refer to the ontology as a whole and cannot be checked for each axiom individually. However, none of these restrictions are relevant for any of the DLs that we consider later in this thesis, so we discuss them only briefly here.

There are two main sources of undecidability in DLs, both due to role compositions. Firstly, unrestricted use of role compositions causes undecidability already in case of fairly inexpressive DLs such as $\mathcal{ALC}$ [19] (with the notable exception of $\mathcal{EL}$; see below). Decidability is recovered by a syntactic *regularity* condition that ensures that role compositions, viewing each $R_1 \circ R_2 \sqsubseteq S$ as a context-free grammar rule $S \rightarrow R_1 R_2$, generate regular languages [32, 56, 61]. Regular role compositions can be eliminated from an ontology without affecting its consequences [59, 118].

The second source of undecidability stems from the interaction of transitive roles with number restrictions, e.g., in the DL $\mathcal{SHIQ}$ [57]. To avoid this interaction, $\mathcal{SHIQ}$ allows number restrictions only with roles that have no transitive subroles. These are called *simple roles*. In $\mathcal{SROIQ}$, the definition of simple roles is adjusted to take general role compositions into account, and non-simple

roles are also forbidden to occur in several other constructs such as local reflexivity concepts and some role characteristics [56].

2.2.2 Light-Weight DLs

Subsumption checking is ExpTime-complete for all DLs between $\mathcal{ALC}$ and $\mathcal{SHIQ}$ [131], and coN2ExpTime-complete for $\mathcal{SROIQ}$ [59]. In recent years, other fragments of DLs have been identified that have more favorable computational properties. This has resulted in three main families of light-weight DLs: $\mathcal{EL}$ [8, 10], DL-Lite [5, 27], and DLP [46], which also correspond respectively to the fragments OWL EL, OWL QL, and OWL RL of the Web Ontology Language.

$\mathcal{EL}$ is a simple tractable fragment of $\mathcal{ALC}$ that restricts concept constructors to top, conjunctions, and existential restrictions [8]. Its extension $\mathcal{EL}^{++}$ further allows bottom, nominals, a restricted form of concrete domains, and complex role inclusions. Subsumption checking in $\mathcal{EL}^{++}$ can still be solved in polynomial time [10], and this is even with arbitrary (non-regular) role compositions. Further tractable extensions of $\mathcal{EL}^{++}$ exist [11, 70]. The $\mathcal{EL}$ family has been used widely to model large but light-weight ontologies that consist mainly of terminological axioms, in particular in the life sciences, such as SNOMED CT [113] and OBO [122].

DL-Lite is a family of DLs that is commonly used in combination with large volumes of data stored in traditional relational databases to augment the expressivity of a query language that retrieves such data. This approach, known as Ontology Based Data Access, considers ontologies as a language for constructing views on top of existing data [101]. The core feature of DL-Lite is that query answering can be realized with standard query languages such as SQL that are not aware of the DL semantics [27]. Ontological information is used only in a preprocessing step for rewriting DL-Lite queries into SQL queries.

DLP is short for *Description Logic Programs* and comprises various DLs that are syntactically restricted in such a way that axioms could also be read as rules in first-order Horn logic without function symbols [46]. For this reason, DLP-type logics can be considered as kinds of rule languages contained in DLs. DLP is often used to augment databases, e.g., in an implementation of OWL RL in the Oracle 11g database management system [67].

2.3 Relationship to OWL

The Web Ontology Language OWL is a knowledge representation language standardized by the World Wide Web Consortium (W3C). The current version of the OWL specification is OWL 2 [100]. OWL is one of the most important applications of DLs today. In this section we briefly outline the relationship of the two languages.

The main building blocks of OWL are very similar to those of DLs, with the main difference being that concepts are called *classes* and roles are called *properties*. Historically, however, OWL has also been conceived as an extension of RDF [84], a Web data modeling language whose expressivity is comparable to DL ABoxes. The formal semantics of RDF [48] is subtly different from that of DLs, even though both lead to the same consequences in many common cases. For this reason, there are two styles of formal semantics for OWL: the *Direct Semantics* [91] based on DLs and the *RDF-Based Semantics* [112].

The Direct Semantics of OWL is only defined for a certain syntactic fragment of OWL called OWL DL. In contrast, the OWL language without any syntactic constraints is called OWL Full; it comprises ontologies that can only be interpreted under the RDF-based Semantics. Under the Direct Semantics, large parts of OWL DL can indeed be considered as a syntactic variant of DLs. For example, the axiom $\mathsf{Mother} \equiv \mathsf{Female} \sqcap \mathsf{Parent}$ would be written as follows in OWL Functional-Style Syntax:

```
EquivalentClasses( Mother ObjectIntersectionOf( Female Parent ) )
```

where the symbols Mother, Female, and Parent would be identifier strings that conform to the OWL specification which is based on Uniform Resource Identifiers (URIs). The above example illustrates the close relationship between the syntax of DLs and that of OWL. In many cases, it is enough to translate an operator symbol of DLs into the corresponding operator name in OWL, which is then written in prefix notation like a function. This is also why the above form of syntax is called *Functional-Style Syntax* [92]. The OWL standard provides a number of syntactic forms that can be used to express OWL ontologies. The most prominent among these is the *RDF/XML Syntax* [20] since it is the only format that all conforming OWL tools need to support. On the other

logic	roles	concepts	axioms	subsumption checking
$\mathcal{EL}^+_\bot$	T	$A, \top, \bot, \sqcap, \exists$	$C \sqsubseteq D, R \sqsubseteq S, R_1 \circ R_2 \sqsubseteq S$	PTIME-complete
$\mathcal{ALCI}$	T, T^-	$A, \top, \bot, \sqcap, \sqcup, \neg, \exists, \forall$	$C \sqsubseteq D$	EXPTIME-complete
$\mathcal{SHI}$	T, T^-	$A, \top, \bot, \sqcap, \sqcup, \neg, \exists, \forall$	$C \sqsubseteq D, R \sqsubseteq S, \mathsf{Tra}(R)$	EXPTIME-complete

Table 2.3: List of the DLs considered in this thesis

hand, it is more difficult for humans to read, and we do not present it here. The OWL API [55], a Java API for manipulating and reasoning with OWL ontologies, provides parsers and writers for all standard OWL syntaxes as well as interfaces for OWL reasoners. Popular reasoners for large parts of OWL DL include FaCT++ [132], HermiT [94], Pellet [121], and RacerPro [47]. Up-to-date lists of current OWL reasoners are best found online.[1]

The expressivity of OWL DL corresponds approximately to the DL $\mathcal{SROIQ}(\mathcal{D})$ [29]. The OWL standard defines three fragments (also known as profiles) [90] that trade expressive power for favorable computation properties. These are closely related to and have similar properties as the three families of light-weight DLs that we discussed above: OWL EL is based on the $\mathcal{EL}$ family of DLs, OWL QL (for Query Language) is based on the DL-Lite family of DLs, and OWL RL (for Rule Language) is based on Description Logic Programs.

OWL also provides a number of non-logical features that are not considered in DLs. These include the ability to give a name and a version to an ontology, import axioms from one ontology into another, declare identifiers, and annotate axioms and entities with extra information, e.g., provenance.

2.4 Roadmap

In this thesis we develop algorithms for the subsumption checking problem and for the ontology classification task. In Part II we work in the $\mathcal{EL}$ family of logics, in particular in the DL $\mathcal{EL}^+_\bot$ which extends $\mathcal{EL}$ with $\bot$ and complex role inclusions. Equivalently, $\mathcal{EL}^+_\bot$ is $\mathcal{EL}^{++}$ without nominals and concrete domains. In Parts III and IV we then move on to the more expressive DL $\mathcal{ALCI}$. However, since it is possible to eliminate role inclusions and transitivity axioms from a $\mathcal{SHI}$ ontology to obtain an $\mathcal{ALCI}$ ontology that entails the same consequences, our results are applicable to the DL $\mathcal{SHI}$ as well. Table 2.3 lists all these DLs. We stress that no syntactic restrictions are necessary for

[1] A list of reasoners can be found, e.g., at `http://semanticweb.org/wiki/Category:Reasoner`.

these logics; each logic allows unrestricted combination of the corresponding features in Table 2.3.

Since our focus is on TBox reasoning, we omit individuals and ABox axioms from the rest of the main presentation. Note that, in all DLs without nominals, the presence of ABox axioms in an ontology can have no effect on the entailment of TBox axioms apart from possibly making the whole ontology inconsistent. Thus, assuming the ontology is consistent, our algorithms can be used for classification even in the presence of an ABox. Moreover, we show in Appendix A that, by treating individuals as atomic concepts, ABox reasoning in $\mathcal{EL}^+_\bot$ can be reduced to TBox reasoning. Unfortunately, this reduction does not work in more expressive DLs, and extending our algorithms to support ABox reasoning in $\mathcal{ALCI}$ and beyond is interesting future work.

Part II

ELK System Description

All proofs inevitably lead to propositions which have no proof! All things are known because we want to believe in them.

—Frank Herbert
Children of Dune

Chapter 3

Consequence-Based Calculus for $\mathcal{EL}^+_\bot$

In this chapter we present the inference system that is implemented in ELK without yet focusing on specific algorithmic details of the implementation. We first formulate a calculus for terminological reasoning in $\mathcal{EL}^+_\bot$ in Section 3.1, and then prove completeness of this calculus in Section 3.2.

3.1 Inference Rules

Our calculus for $\mathcal{EL}^+_\bot$ is based on a set of inference rules that are similar to the completion rules originally proposed for $\mathcal{EL}^{++}$ [10]. The improvement and extension of these rules to more expressive logics is a part of the ongoing development of ELK [63, 64, 65].

Given an $\mathcal{EL}^+_\bot$ ontology $\mathcal{O}$, we write $\sqsubseteq^*_\mathcal{O}$ for the smallest reflexive transitive binary relation over roles such that $R \sqsubseteq^*_\mathcal{O} S$ holds for all $R \sqsubseteq S \in \mathcal{O}$. We say that a concept C *occurs negatively* (respectively *positively*) in an ontology $\mathcal{O}$, if C is a syntactic subexpression of D (respectively E) for some concept inclusion $D \sqsubseteq E \in \mathcal{O}$.

The inference rules for $\mathcal{EL}^+_\bot$ are shown in Fig. 3.1. They operate with expressions of the form $\mathsf{init}(C)$, $C \sqsubseteq D$, $C \xrightarrow{R} D$, where C and D are (possibly complex) concepts. We call the expression $C \xrightarrow{R} D$ an (existential) *link*, and expression $\mathsf{init}(C)$ a *concept initialization*. Intuitively, $\mathsf{init}(C)$ is used to initialize derivation of subsumers for C, and expression $C \xrightarrow{R} D$ means that for two (initialized) concepts C and D, the subsumption $C \sqsubseteq \exists R.D$ is entailed. We distinguish between the premises of a rule (appearing above the horizontal line) and its side conditions (appearing after the colon). Note that the subsumptions in $\mathcal{O}$ are only used as side conditions of rule $\mathbf{R}_\sqsubseteq$; to distinguish them from the

$$\mathbf{R_0}\ \frac{\mathsf{init}(C)}{C \sqsubseteq C} \qquad \mathbf{R}_\top\ \frac{\mathsf{init}(C)}{C \sqsubseteq \top} : \top \text{ occurs negatively in } \mathcal{O} \qquad \mathbf{R}_\bot\ \frac{E \xrightarrow{R} C \quad C \sqsubseteq \bot}{E \sqsubseteq \bot}$$

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

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

$$\mathbf{R}_\sqsubseteq\ \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O} \qquad \mathbf{R}_\circ\ \frac{E \xrightarrow{R_1} C \quad C \xrightarrow{R_2} D}{E \xrightarrow{S} D} : \begin{array}{l} S_1 \circ S_2 \sqsubseteq S \in \mathcal{O} \\ R_1 \sqsubseteq^*_\mathcal{O} S_1 \\ R_2 \sqsubseteq^*_\mathcal{O} S_2 \end{array}$$

Figure 3.1: Inference rules for reasoning in $\mathcal{EL}^+_\bot$

derived subsumptions, we refer to them as *told subsumptions*.

The inference procedure starts with a set Input of expressions $\mathsf{init}(C)$ for concepts C for which computation of subsumers is required. Intuitively, the subsumers are computed by: (*i*) deriving trivial subsumers using rules $\mathbf{R_0}$ and $\mathbf{R}_\top$, (*ii*) combining subsumers into complex subsumers using *composition rules* $\mathbf{R}^+_\sqcap$ and $\mathbf{R}^+_\exists$, (*iii*) expanding subsumers under told subsumptions using $\mathbf{R}_\sqsubseteq$, and (*iv*) decomposing complex subsumers using *decomposition rules* $\mathbf{R}^-_\sqcap$ and $\mathbf{R}^-_\exists$.

The mechanism of composition and decomposition rules for conjunction $\mathbf{R}^+_\sqcap$ and $\mathbf{R}^-_\sqcap$ is relatively straightforward. The rules $\mathbf{R}^+_\exists$ and $\mathbf{R}^-_\exists$ for existential restrictions involve auxiliary existential links. Intuitively, whenever a subsumption $C \sqsubseteq \exists R.D$ with an existential restriction is derived, the decomposition rule $\mathbf{R}^-_\exists$ produces a link $C \xrightarrow{R} D$ and initializes the derivation of subsumers for D. The link together with the derived subsumers for D can then be used to derive new existential subsumers for the original concept C using the composition rule $\mathbf{R}^+_\exists$. Apart from deriving new existential subsumers, existential links can also be used by rule $\mathbf{R}_\bot$ for propagation of inconsistency and by rule $\mathbf{R}_\circ$ for composing links using role compositions.

Example 3.1. Consider the ontology $\mathcal{O}$ consisting of the following axioms.

$$A \sqsubseteq \exists R.(C \sqcap D) \tag{3.1}$$

$$B \equiv A \sqcap \exists S.D \tag{3.2}$$

$$\exists S.D \sqsubseteq C \tag{3.3}$$

$$R \sqsubseteq S \tag{3.4}$$

To compute the subsumers of A, we compute the closure of $\mathsf{Input} = \{\mathsf{init}(A)\}$ under the inference rules in Fig. 3.1.

$\mathsf{init}(A)$	input expression	(3.5)
$A \sqsubseteq A$	by $\mathbf{R_0}$ to (3.5)	(3.6)
$A \sqsubseteq \exists R.(C \sqcap D)$	by $\mathbf{R}_{\sqsubseteq}$ to (3.6) using (3.1)	(3.7)
$\mathsf{init}(C \sqcap D)$	by $\mathbf{R}^-_{\exists}$ to (3.7)	(3.8)
$A \xrightarrow{R} C \sqcap D$	by $\mathbf{R}^-_{\exists}$ to (3.7)	(3.9)
$C \sqcap D \sqsubseteq C \sqcap D$	by $\mathbf{R_0}$ to (3.8)	(3.10)
$C \sqcap D \sqsubseteq C$	by $\mathbf{R}^-_{\sqcap}$ to (3.10)	(3.11)
$C \sqcap D \sqsubseteq D$	by $\mathbf{R}^-_{\sqcap}$ to (3.10)	(3.12)
$A \sqsubseteq \exists S.D$	by $\mathbf{R}^+_{\exists}$ to (3.9) and (3.12) using (3.4)	(3.13)
$A \sqsubseteq A \sqcap \exists S.D$	by $\mathbf{R}^+_{\sqcap}$ to (3.6) and (3.13)	(3.14)
$\mathsf{init}(D)$	by $\mathbf{R}^-_{\exists}$ to (3.13)	(3.15)
$A \xrightarrow{S} D$	by $\mathbf{R}^-_{\exists}$ to (3.13)	(3.16)
$A \sqsubseteq C$	by $\mathbf{R}_{\sqsubseteq}$ to (3.13) using (3.3)	(3.17)
$A \sqsubseteq B$	by $\mathbf{R}_{\sqsubseteq}$ to (3.14) using (3.2)	(3.18)
$D \sqsubseteq D$	by $\mathbf{R_0}$ to (3.15)	(3.19)

The application of rules $\mathbf{R}^+_{\exists}$ and $\mathbf{R}^+_{\sqcap}$ in lines (3.13) and (3.14) uses the fact that 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 (3.2). Intuitively, these rules are used to gradually construct the subsumption $A \sqsubseteq A \sqcap \exists S.C$, so that rule $\mathbf{R}_{\sqsubseteq}$ with side condition (3.2) can be applied to derive $A \sqsubseteq B$.

From (3.6), (3.13), (3.14), (3.17), (3.18) 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.

What can be said about the subsumptions derived by the inference rules? We can show that all of them are entailed by the ontology $\mathcal{O}$, that is, the inference system is *sound*. To prove this fact, we extend the interpretation relation to expressions $\mathsf{init}(C)$ and $C \xrightarrow{R} D$ by defining that $\mathcal{I} \models \mathsf{init}(C)$

always holds and that $\mathcal{I} \models C \xrightarrow{R} D$ if and only if $\mathcal{I} \models C \sqsubseteq \exists R.D$. Then it is easy to see for each inference rule in Fig. 3.1 that, if $\mathcal{O}$ entails all premises of the rule, then $\mathcal{O}$ entails all of its conclusions. Since, by our definition, $\mathcal{O}$ entails all expressions in Input, one can conclude that $\mathcal{O}$ entails every expression derivable from Input.

The converse of this property does not necessarily hold. That is, the inference system is not complete for deriving all entailed subsumptions with complex concepts. Indeed, from Example 3.1 it is easy to see that $\mathcal{O} \models A \sqsubseteq B \sqcap C$ (because $\mathcal{O} \models A \sqsubseteq B$ and $\mathcal{O} \models A \sqsubseteq C$), but $A \sqsubseteq B \sqcap C$ is not derivable. However, as described in the following theorem, the system is complete for deriving atomic subsumers of initialized concepts. The theorem will be proved in the next section.

Theorem 3.2 (Completeness)**.** *Let $\mathcal{O}$ be an $\mathcal{EL}^+_\bot$ ontology, let* Input *be a set of expressions* $\mathsf{init}(C)$*, and let* Closure *be the closure of* Input *under the inference rules in Fig. 3.1 w.r.t. $\mathcal{O}$. Then, for each concept C such that* $\mathsf{init}(C) \in$ Closure *and $C \sqsubseteq \bot \notin$* Closure*, each atomic concept A, and each atomic role S, we have*

(i) C is satisfiable w.r.t. $\mathcal{O}$;

(ii) if $\mathcal{O} \models C \sqsubseteq A$, then $C \sqsubseteq A \in$ Closure*;*

(iii) if $\mathcal{O} \models C \sqsubseteq \exists S.A$, then there exist R and D such that $C \xrightarrow{R} D \in$ Closure*, $R \sqsubseteq^*_{\mathcal{O}} S$, and $D \sqsubseteq A \in$* Closure*.*

For example, for checking consistency of an ontology $\mathcal{O}$, it is sufficient to verify whether $\top \sqsubseteq \bot$ can be derived from Input containing $\mathsf{init}(\top)$. The rules can also be used for 'one pass' ontology classification, which can be accomplished by computing the closure of Input containing $\mathsf{init}(A)$ for every atomic concept A occurring in $\mathcal{O}$ and considering only derived subsumptions with atomic concepts or $\bot$.

To estimate the complexity of the procedure, note that the inference rules can only derive expressions of the form $C \sqsubseteq D$, $C \xrightarrow{R} D$, and $\mathsf{init}(C)$ where C, D, and R occur in $\mathcal{O} \cup$ Input, and each rule has a bounded number of premises and side conditions. Therefore, the closure under the inference rules can be computed in time polynomial in the size of $\mathcal{O}$ and Input.

3.2 Canonical Models

A commonly used technique for proving completeness of $\mathcal{EL}$ algorithms is based on canonical model construction.[1] Similarly to saturation-based theorem proving [18], canonical models are constructed (uniquely) from the closure of a set of expressions under inference rules. The canonical model not only witnesses satisfiability of the input ontology, but can also be used to refute all non-subsumptions with initialized concepts.

Assume that $\mathcal{O}$, Input, and Closure satisfy the condition of Theorem 3.2. If $C \sqsubseteq \bot \in \mathsf{Closure}$ for every concept C such that $\mathsf{init}(C) \in \mathsf{Closure}$, then Theorem 3.2 trivially holds. Therefore, in the rest of this section we can assume that we have $\mathsf{init}(C) \in \mathsf{Closure}$ and $C \sqsubseteq \bot \notin \mathsf{Closure}$ for at least one concept C. In this case, one can define the canonical model (of $\mathcal{O}$ w.r.t. Closure) as follows.

Definition 3.3 (Canonical Model)**.** The *canonical model* $\mathcal{I}$ is defined by

$$\Delta^{\mathcal{I}} = \{x_C \mid \mathsf{init}(C) \in \mathsf{Closure} \text{ and } C \sqsubseteq \bot \notin \mathsf{Closure}\},$$

$$A^{\mathcal{I}} = \{x_C \in \Delta^{\mathcal{I}} \mid C \sqsubseteq A \in \mathsf{Closure}\},$$

$$S^{\mathcal{I}} = \{\langle x_C, x_D\rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \text{there exists } R \text{ such that } C \xrightarrow{R} D \in \mathsf{Closure} \text{ and } R \sqsubseteq^*_{\mathcal{O}} S\}.$$

The domain $\Delta^{\mathcal{I}}$ contains a distinct element x_C for each concept C such that $\mathsf{init}(C) \in \mathsf{Closure}$ and $C \sqsubseteq \bot \notin \mathsf{Closure}$. Since there is at least one such C by the assumption, the domain is nonempty.

Example 3.4. Consider the set of expressions (3.5)–(3.19), which were computed in Example 3.1. Then the canonical model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ w.r.t. (3.5)–(3.19) is defined by:

- $\Delta^{\mathcal{I}} = \{x_A,\ x_{C\sqcap D},\ x_D\}$,
- $A^{\mathcal{I}} = B^{\mathcal{I}} = \{x_A\}$, $C^{\mathcal{I}} = \{x_A,\ x_{C\sqcap D}\}$, $D^{\mathcal{I}} = \{x_{C\sqcap D},\ x_D\}$,
- $R^{\mathcal{I}} = \{\langle x_A, x_{C\sqcap D}\rangle\}$, $S^{\mathcal{I}} = \{\langle x_A, x_{C\sqcap D}\rangle,\ \langle x_A, x_D\rangle\}$.

It is easy to see that $\mathcal{I}$ is a model of axioms (3.1)–(3.4).

Of course, even if $\mathcal{I}$ is well-defined, it does not immediately follow that it is a model of $\mathcal{O}$. In order to prove that this is indeed the case, we first need two auxiliary lemmas.

[1] In the earlier $\mathcal{EL}$ papers [8, 10, 11], canonical models were called *completion graphs*.

Lemma 3.5. *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 prove that $x_C \in D^{\mathcal{I}}$.

- Case $D = A$: $x_C \in A^{\mathcal{I}}$ follows from our assumptions and 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 $D = \bot$: This case cannot occur, since $x_C \in \Delta^{\mathcal{I}}$ implies $C \sqsubseteq \bot \notin \mathsf{Closure}$.
- Case $D = D_1 \sqcap D_2$: Since $\mathsf{Closure}$ is closed under $\mathbf{R}^{-}_{\sqcap}$, we have $C \sqsubseteq D_i \in \mathsf{Closure}$ for $i = 1, 2$. Therefore, 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 $D = \exists S.D_2$: Since $\mathsf{Closure}$ is closed under $\mathbf{R}^{-}_{\exists}$, we have $\mathsf{init}(D_2) \in \mathsf{Closure}$ and $C \xrightarrow{S} D_2 \in \mathsf{Closure}$. Now $x_C \in \Delta^{\mathcal{I}}$ implies $C \sqsubseteq \bot \notin \mathsf{Closure}$, so, since $C \xrightarrow{S} D_2 \in \mathsf{Closure}$, $D_2 \sqsubseteq \bot \notin \mathsf{Closure}$ follows due to $\mathbf{R}_{\bot}$. Thus $x_{D_2} \in \Delta^{\mathcal{I}}$. Also, $\mathsf{init}(D_2) \in \mathsf{Closure}$ implies $D_2 \sqsubseteq D_2 \in \mathsf{Closure}$ due to rule $\mathbf{R_0}$. By the induction hypothesis applied to $x_{D_2} \in \Delta^{\mathcal{I}}$ and $D_2 \sqsubseteq D_2 \in \mathsf{Closure}$ we get $x_{D_2} \in D_2^{\mathcal{I}}$. Since $C \xrightarrow{S} D_2 \in \mathsf{Closure}$, we have $\langle x_C, x_{D_2} \rangle \in S^{\mathcal{I}}$ by the definition of $S^{\mathcal{I}}$. Then $x_C \in (\exists S.D_2)^{\mathcal{I}}$ holds by the semantics of $\exists$. □

Corollary 3.6. *For each C such that $\mathsf{init}(C) \in \mathsf{Closure}$ and $C \sqsubseteq \bot \notin \mathsf{Closure}$, we have $x_C \in C^{\mathcal{I}}$.*

Proof. Suppose $\mathsf{init}(C) \in \mathsf{Closure}$ and $C \sqsubseteq \bot \notin \mathsf{Closure}$. Then $x_C \in \Delta^{\mathcal{I}}$ by the definition of $\Delta^{\mathcal{I}}$, and $C \sqsubseteq C$ due to rule $\mathbf{R_0}$. Hence $x_C \in C^{\mathcal{I}}$ follows by Lemma 3.5. □

The next lemma is the converse of Lemma 3.5 for D occurring negatively in $\mathcal{O}$.

Lemma 3.7. *For each $x_C \in \Delta^{\mathcal{I}}$ and each D occurring negatively in $\mathcal{O}$, $x_C \in D^{\mathcal{I}}$ implies $C \sqsubseteq D \in \mathsf{Closure}$.*

Proof. The proof is by structural induction on D. In each case, we assume that D occurs negatively in $\mathcal{O}$ and $x_C \in D^{\mathcal{I}}$, and prove that $C \sqsubseteq D \in \mathsf{Closure}$.

- Case $D = A$: If $x_C \in A^{\mathcal{I}}$, then $C \sqsubseteq A \in \mathsf{Closure}$ by the definition of $A^{\mathcal{I}}$.

- Case $D = \top$: By assumption, $\top$ occurs negatively in $\mathcal{O}$ and $x_C \in \top^{\mathcal{I}} = \Delta^{\mathcal{I}}$. Then $\mathsf{init}(C) \in \mathsf{Closure}$ by the definition of $\Delta^{\mathcal{I}}$, from which $C \sqsubseteq \top \in \mathsf{Closure}$ follows since $\mathsf{Closure}$ is closed under rule $\mathbf{R}_{\top}$.

- 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 assumption, $D_1 \sqcap D_2$ occurs negatively in $\mathcal{O}$ and $x_C \in (D_1 \sqcap D_2)^{\mathcal{I}}$. Then $x_C \in D_i^{\mathcal{I}}$ by the semantics of $\sqcap$ and D_i occurs negatively in $\mathcal{O}$ for $i = 1, 2$. Applying the induction hypothesis to $x_C \in D_i^{\mathcal{I}}$ we obtain $C \sqsubseteq D_i \in \mathsf{Closure}$, from which $C \sqsubseteq D_1 \sqcap D_2 \in \mathsf{Closure}$ follows since $\mathsf{Closure}$ is closed under rule $\mathbf{R}^{+}_{\sqcap}$ and $D_1 \sqcap D_2$ occurs negatively in $\mathcal{O}$.

- Case $D = \exists S.D_2$: Since $\exists S.D_2$ occurs negatively in $\mathcal{O}$ and $x_C \in (\exists S.D_2)^{\mathcal{I}}$ by assumption, there exists some element $x_{D_1} \in \Delta^{\mathcal{I}}$ such that $\langle x_C, x_{D_1} \rangle \in S^{\mathcal{I}}$ and $x_{D_1} \in D_2^{\mathcal{I}}$ by the semantics of $\exists$, and D_2 occurs negatively in $\mathcal{O}$. Applying the induction hypothesis to $x_{D_1} \in D_2^{\mathcal{I}}$ we obtain $D_1 \sqsubseteq D_2 \in \mathsf{Closure}$. Further, by the definition of $S^{\mathcal{I}}$, there exists R such that $C \xrightarrow{R} D_1 \in \mathsf{Closure}$ and $R \sqsubseteq^{*}_{\mathcal{O}} S$. Since $\mathsf{Closure}$ is closed under rule $\mathbf{R}^{+}_{\exists}$ and the rule is applicable to premises $C \xrightarrow{R} D_1$ and $D_1 \sqsubseteq D_2$ with side conditions that $\exists S.D_2$ occurs negatively in $\mathcal{O}$ and $R \sqsubseteq^{*}_{\mathcal{O}} S$, we conclude $C \sqsubseteq \exists S.D_2 \in \mathsf{Closure}$. □

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

Theorem 3.8. $\mathcal{I} \models \mathcal{O}$.

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

- Case $\alpha = D \sqsubseteq E$: We consider an arbitrary element $x_C \in D^{\mathcal{I}}$ and show that $x_C \in E^{\mathcal{I}}$. By Lemma 3.7, since D occurs negatively in $\mathcal{O}$ and $x_C \in D^{\mathcal{I}}$, we have $C \sqsubseteq D \in \mathsf{Closure}$. Then, since $\mathsf{Closure}$ is closed under rule $\mathbf{R}_{\sqsubseteq}$, we have $C \sqsubseteq E \in \mathsf{Closure}$, from which $x_C \in E^{\mathcal{I}}$ follows by Lemma 3.5. Therefore $\mathcal{I} \models D \sqsubseteq E$.

- Case $\alpha = S_1 \sqsubseteq S_2$: We consider arbitrary elements $\langle x_C, x_D \rangle \in S_1^{\mathcal{I}}$ and show that $\langle x_C, x_D \rangle \in S_2^{\mathcal{I}}$. By the definition of $S_1^{\mathcal{I}}$, there exists R such that $C \xrightarrow{R} D \in \mathsf{Closure}$ and $R \sqsubseteq^{*}_{\mathcal{O}} S_1$. The latter and $S_1 \sqsubseteq S_2 \in \mathcal{O}$ implies $R \sqsubseteq^{*}_{\mathcal{O}} S_2$, so $\langle x_C, x_D \rangle \in S_2^{\mathcal{I}}$ by the definition of S_2. Therefore $\mathcal{I} \models S_1 \sqsubseteq S_2$.

- Case $\alpha = S_1 \circ S_2 \sqsubseteq S$: We consider arbitrary $\langle x_C, x_D \rangle \in S_1^{\mathcal{I}}$ and $\langle x_D, x_E \rangle \in S_2^{\mathcal{I}}$ and show that $\langle x_C, x_E \rangle \in S^{\mathcal{I}}$. By the definition of $S_1^{\mathcal{I}}$ and $S_2^{\mathcal{I}}$, there exist R_1 and R_2 such that $C \xrightarrow{R_1} D \in \mathsf{Closure}$, $D \xrightarrow{R_2} E \in \mathsf{Closure}$, $R_1 \sqsubseteq_{\mathcal{O}}^* S_1$, and $R_2 \sqsubseteq_{\mathcal{O}}^* S_2$. Then, since $\mathsf{Closure}$ is closed under rule $\mathbf{R}_\circ$, we have $C \xrightarrow{S} E \in \mathsf{Closure}$, from which $\langle x_C, x_E \rangle \in S^{\mathcal{I}}$ follows by the definition of S. Therefore $\mathcal{I} \models S_1 \circ S_2 \sqsubseteq S$. □

It is now easy to conclude the proof of completeness.

Proof of Theorem 3.2. Consider an arbitrary concept C such that $\mathsf{init}(C) \in \mathsf{Closure}$ and $C \sqsubseteq \bot \notin \mathsf{Closure}$, an arbitrary atomic concept A, and an arbitrary atomic role S. Let $\mathcal{I}$ be the canonical model of $\mathcal{O}$ w.r.t. $\mathsf{Closure}$. By Corollary 3.6, we have $x_C \in C^{\mathcal{I}}$.

(i) C is satisfiable w.r.t. $\mathcal{O}$ since $x_C \in C^{\mathcal{I}}$ implies $C^{\mathcal{I}} \neq \emptyset$.

(ii) If $\mathcal{O} \models C \sqsubseteq A$, then $x_C \in A^{\mathcal{I}}$, hence the required $C \sqsubseteq A \in \mathsf{Closure}$ follows from the definition of $A^{\mathcal{I}}$.

(iii) If $\mathcal{O} \models C \sqsubseteq \exists S.A$, then $x_C \in (\exists S.A)^{\mathcal{I}}$, so there exists an element $x_D \in \Delta^{\mathcal{I}}$ with $\langle x_C, x_D \rangle \in S^{\mathcal{I}}$ and $x_D \in A^{\mathcal{I}}$; hence, by the definition of $S^{\mathcal{I}}$ and $A^{\mathcal{I}}$, there exists some R such that $C \xrightarrow{R} D \in \mathsf{Closure}$, $R \sqsubseteq_{\mathcal{O}}^* S$, and $D \sqsubseteq A \in \mathsf{Closure}$. □

What do such machines really do? They increase the number of things we can do without thinking. Things we do without thinking—there's the real danger.

—Frank Herbert
God Emperor of Dune

Chapter 4

Saturation Procedures

In the previous chapter we discussed soundness and completeness of our calculus for $\mathcal{EL}^+_\bot$. In this chapter 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 difference in performance.

We first give a high-level overview of the procedure in Section 4.1, which can essentially be applied to any inference system, and provide a more detailed description that is specific to $\mathcal{EL}^+_\bot$ in Section 4.2. We then present concurrent extensions of both procedures in Sections 4.3 and 4.4.

4.1 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 procedure is well-known: it is similar to the 'given clause' algorithm (set of support strategy) used in saturation-based theorem proving (see, e.g., [18, 139]) and semi-naive (bottom-up) evaluation of logic programs (see, e.g., [1]).

The abstract saturation procedure can be described using Algorithm 4.1. In order to compute the deductive closure for a set Input of expressions under inference rules, the procedure maintains two collections of expressions: the queue Todo containing expressions to which the rules have yet to be applied and the set Closure that accumulates the result of the computation. The queue Todo

Algorithm 4.1: The abstract saturation procedure

```
1 for expression ∈ Input do
2     Todo.add(expression);
3 while (expression ← Todo.poll()) ≠ null do
4     process(expression);
5 process(expression) :
6     if Closure.add(expression) then
7         for inference ∈ inferences(expression, Closure) do
8             Todo.add(inference.conclusion)
```

is initialized with expressions from Input (lines 1–2), and every expression in Todo is processed by function `process`(expression) until the queue is empty (lines 3–4). When the expression is processed, the procedure first attempts to insert it into Closure (line 6). If this operation is successful, i.e., the expression was not in Closure, then all inferences between this expression and the expressions in Closure are applied and the conclusions of the inferences are added to the Todo queue (lines 7–8).

Example 4.1. The derivation in Example 3.1 already presents the expressions in the order in which they are processed by the abstract saturation procedure. For example, after processing expression (3.10), Closure contains expressions (3.5)–(3.10), and Todo contains expressions (3.11) and (3.12). The algorithm then takes and removes expression (3.11) from Todo, adds it to Closure, and applies all inferences involving this expression and the previously processed expressions (3.5)–(3.10). In this case, no inference rules are applicable. The algorithm then takes the next expression (3.12) from Todo, adds it to Closure, and applies all inferences involving this expression and the previously processed expressions (3.5)–(3.11). In this case, rule $\mathbf{R}^{+}_{\exists}$ is applicable to the premises (3.9) and (3.12), so the conclusion (3.13) is added to Todo.

Correctness of Algorithm 4.1 is a consequence of the following (semi-)invariants that can be proved by induction over execution of the algorithm:

(1) Every expression in Todo and Closure either occurs in Input or is obtained by an inference rule from some expressions in Closure;

(2) Closure does not contain duplicate expressions;

(3) After every iteration of the loop at lines 3–4:

(a) Either the size of Closure increases or, otherwise, it remains the same and the size of Todo decreases;

(b) every element in Input and every conclusion of an inference from Closure occurs either in Todo or in Closure.

From (1), it follows that Algorithm 4.1 can add to Closure only expressions that are derivable from Input. Therefore, from (2) and (3a), it follows that if there are only finitely many different expressions that can be derived from Input, then Algorithm 4.1 terminates. Finally, from (3b) it follows that when Algorithm 4.1 terminates (and thus Todo is empty), Closure contains all the expressions that are derivable from Input.

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

One of the key details of Algorithm 4.1 that has not been discussed so far is the application of inference rules in line 7. At this point, the abstract saturation procedure requires computing all inferences between the given expression and the expressions in Closure. How to implement this computation? One possibility is to enumerate all tuples of expressions in Closure that contain the given expression (note that the given expression will already be present in Closure at this point) and check application of all rules to each tuple. Although this operation can be implemented in polynomial time in the size of Closure, this procedure will, clearly, be impractical. Instead, modern $\mathcal{EL}$ reasoners use sophisticated control structures that enable more efficient search and application of inference rules (see, e.g., [14] for a description of the CEL reasoner).

In this section we describe the algorithms and datastructures used in ELK. Our optimized saturation procedure for $\mathcal{EL}^+_\bot$ consists of three phases: the first two can be seen as a kind of compilation for rules which enables their efficient application in the third phase.

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. 3.1 can be efficiently verified. For this purpose, we store information about occurrences of roles and concepts in $\mathcal{O}$ using a system of *indexed objects*.

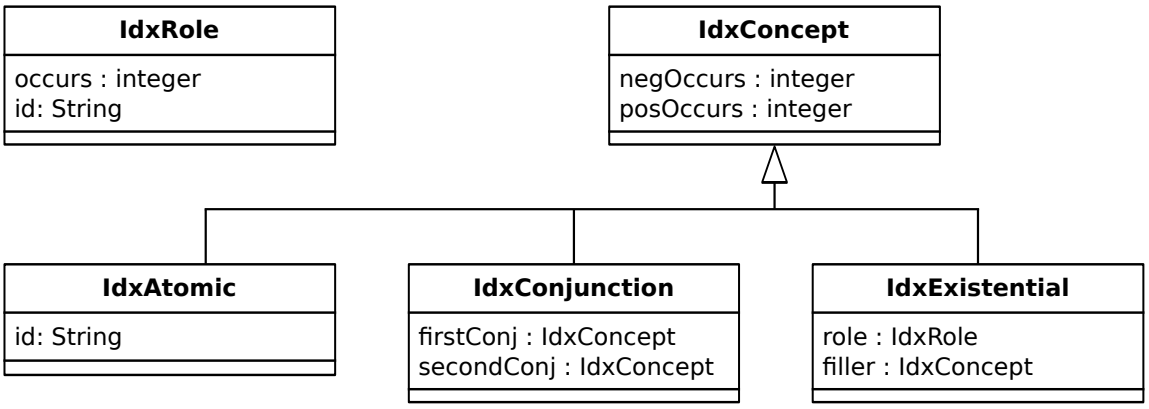


Figure 4.1: Classes for indexing objects

The type hierarchy of indexed objects is shown in Fig. 4.1; it closely follows the recursive definition of $\mathcal{EL}$ expressions. The top level classes IdxRole and IdxConcept are used to 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. The different types of $\mathcal{EL}$ concepts are represented by the corresponding subclasses of IdxConcept from Fig. 4.1. For example, a conjunction $C \sqcap D$ is represented by an object of type IdxConjunction, and its fields firstConj and secondConj point to the indexed objects that represent the concepts C and D, respectively. In addition, there are distinguished instances top and bottom of IdxConcept that represent the concepts $\top$ 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.

$$\mathsf{negConjs} = \{\langle C, D, C \sqcap D\rangle \mid C \sqcap D \text{ occurs negatively in } \mathcal{O}\}$$

$$\mathsf{negExists} = \{\langle R, C, \exists R.C\rangle \mid \exists R.C \text{ occurs negatively in } \mathcal{O}\}$$

$$\mathsf{concIncs} = \{\langle C, D\rangle \mid C \sqsubseteq D \in \mathcal{O}\}$$

$$\mathsf{roleIncs} = \{\langle R, S\rangle \mid R \sqsubseteq S \in \mathcal{O}\}$$

$$\mathsf{roleComps} = \{\langle R_1, R_2, S\rangle \mid R_1 \circ R_2 \sqsubseteq S \in \mathcal{O}\}$$

These data structures provide a complete representation of $\mathcal{O}$.

Example 4.2. Consider the ontology $\mathcal{O}$ from Example 3.1. Its index contains the following indexed objects with occurrence counts as indicated.[1]

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

The look-up tables contain the following tuples.

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

To give an idea how these index datastructures can be used during the saturation phase, consider the point in the derivation from Example 3.1 when the saturation algorithm processes the expression $A \sqsubseteq \exists S.D$ in line (3.13). To apply rule $\mathbf{R}^+_\sqcap$ 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 of satisfying the side condition. Since negConjs contains $\langle A, \exists S.D, A \sqcap \exists S.D\rangle$, the algorithm checks if the set of processed expressions contains $A \sqsubseteq A$, which can be used as the first premise of $\mathbf{R}^+_\sqcap$. 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 the 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

[1] Recall that we treat the equivalence axiom (3.2) as two individual concept inclusion axioms, so, e.g., the occurrence count of S is 4, not 3.

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 4.3. Let us see how the index data structures in Example 4.2 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 (3.3)), 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 $\mathbf{R}^+_\exists$ and $\mathbf{R}_\circ$, 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.

Example 4.4. For the table roleIncs in Example 4.2 we have $\mathsf{hier} = \{\langle R, R\rangle, \langle R, S\rangle, \langle S, S\rangle\}$.

4.2.3 Phase 3: Saturation of Concepts

In this phase we compute the saturation of Input under the inference rules in Fig. 3.1 using a specialized version of Algorithm 4.1. Recall that our inference rules operate with three types of expressions $\mathsf{init}(C)$, $C \sqsubseteq D$, and $E \xrightarrow{R} C$. Depending on where these expressions are saved, we use different representations for them. The expressions in Todo are represented by objects with the corresponding

$\mathbf{R_0}$: If inits(C),
then subs(C, C).

$\mathbf{R}_\top$: If inits(C) & top.negOccurs > 0,
then subs(C, top).

$\mathbf{R}_\bot$: If links(E, R, C) & subs(C, bottom),
then subs(E, bottom).

$\mathbf{R}^-_\sqcap$: If subs(C, D) & D **instanceOf** IdxConjunction,
then subs(C, D.firstConj) and subs(C, D.secondConj).

$\mathbf{R}^+_\sqcap$: If subs(C, D_1) & subs(C, D_2) & negConjs(D_1, D_2, D),
then subs(C, D).

$\mathbf{R}^-_\exists$: If subs(C, D) & D **instanceOf** IdxExistential,
then inits(D.filler) and links(C, D.role, D.filler).

$\mathbf{R}^+_\exists$: If links(E, R, C) & subs(C, D) & negExists(S, D, F) & hier(R, S),
then subs(E, F).

$\mathbf{R}_\sqsubseteq$: If subs(C, D) & conclncs(D, E),
then subs(C, E).

$\mathbf{R}_\circ$: If links(E, R_1, C) & links(C, R_2, D) & roleComps(S_1, S_2, S) & hier(R_1, S_1) & hier(R_2, S_2),
then links(E, S, D).

Figure 4.2: The closure properties induced by rules in Fig. 3.1

number of parameters, whereas the expressions in Closure are represented using three tables inits, subs, and links for the respective types of expressions as below:

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
$E \xrightarrow{R} C$	Link(E,R,C)	$\langle E, R, C\rangle \in$ links

Before presenting details of the saturation algorithm, we first reformulate in Fig. 4.2 the rules from Fig. 3.1 as closure properties in the above representation. In these rule formulations, for each table T and a tuple x the expression $T(x)$ stands for $x \in T$.

The main body of the optimized saturation algorithm remains the same as for the abstract saturation procedure (Algorithm 4.1, lines 1–4). We only replace the implementation of the function `process`(expression) as shown in Algorithm 4.2. Processing of each expression is done according to its type, and corresponds to all possible ways the expression can be used as a premise of the rules in Fig. 4.2. Note that Algorithm 4.2 uses a number of iterations over joins of tables; we discuss possible ways of optimizing these iterations in Section 5.4.

Algorithm 4.2: process(expression)

```
1  process(Init(C)) :
2      if inits.add(C) then
3          Todo.add(new Sub(C, C));                                   // rule R_0
4          if top.negOccurs > 0 then
5              Todo.add(new Sub(C, top));                             // rule R_⊤
6  process(Sub(C, D)) :
7      if subs.add(⟨C, D⟩) then
8          if D = bottom then
9              for each E, R with links(E, R, C) do
10                 Todo.add(new Sub(E, bottom));                      // rule R_⊥
11         if D instanceOf IdxConjunction then
12             Todo.add(new Sub(C, D.firstConj));
13             Todo.add(new Sub(C, D.secondConj));                    // rule R⁻_⊓
14         for each D_2, E with subs(C, D_2) and negConjs(D, D_2, E) do
15             Todo.add(new Sub(C, E));                               // rule R⁺_⊓
16         for each D_1, E with subs(C, D_1) and negConjs(D_1, D, E) do
17             Todo.add(new Sub(C, E));                               // rule R⁺_⊓
18         if D instanceOf IdxExistential then
19             Todo.add(new Init(D.filler));
20             Todo.add(new Link(C, D.role, D.filler));               // rule R⁻_∃
21         for each E, F, R, S with links(E, R, C) and negExists(S, D, F) and hier(R, S) do
22             Todo.add(new Sub(E, F));                               // rule R⁺_∃
23         for each E with concIncs(D, E) do
24             Todo.add(new Sub(C, E));                               // rule R_⊑
25 process(Link(E, R, C)) :
26     if links.add(⟨E, R, C⟩) then
27         if subs(C, bottom) then
28             Todo.add(new Sub(E, bottom));                          // rule R_⊥
29         for each D, F, S with subs(C, D) and negExists(S, D, F) and hier(R, S) do
30             Todo.add(new Sub(E, F));                               // rule R⁺_∃
31         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
32             Todo.add(new Link(E, S, D));                           // rule R_∘
33         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
34             Todo.add(new Link(D, S, C));                           // rule R_∘
```

4.3 Abstract Concurrent Saturation Procedure

One surprising property of the abstract saturation procedure described in Section 4.1 is that it works correctly in the concurrent setting. Specifically, the while loop 3–4 of Algorithm 4.1 can be executed from several independent 'workers' running in parallel, that repeatedly take the next expression from the shared Todo queue and perform inferences between this expression and the expressions in the shared set Closure. Concurrent processing of inferences can considerably accelerate the saturation process on multiprocessor systems. Of course, the implementations of Todo and Closure should support concurrent operations, such as addition, removal, and iteration over elements. In particular, when performing inferences between the given expression and Closure (lines 7–8), at least all inferences with the expressions that have occurred in Closure at the beginning of the iteration should be computed, even if new expressions are inserted by other workers during the iteration. This property is sufficient to guarantee that no inferences between expressions in Closure get lost: if an inference between some tuple of expressions in Closure is possible, it will be applied by the worker that inserts the last expression from this tuple into Closure since all other expressions will already occur in Closure when this worker starts its iteration over the inferences.

However, concurrent (thread-safe) datastructures satisfying the above requirements might not always be available, or may exhibit certain overheads compared to conventional datastructures. In this section we describe an alternative concurrent method for saturation which uses just concurrent queues and Booleans with atomic 'compare and swap' operations. These features are more common in programming libraries. 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 set of *contexts* and a function `getContexts`(expression) assigning to every expression a nonempty subset of contexts such that, whenever an inference between several expressions is possible, at least one common context is assigned to these expressions. We will present a concrete context assignment in Section 4.4.

The new concurrent saturation procedure is described in Algorithm 4.3. The procedure works as follows. 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 4.1, but they only contain expressions to which this context is assigned. In addition, the procedure maintains a queue activeContexts to keep

Algorithm 4.3: The abstract concurrent saturation procedure

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

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 the expression into c.Todo and *activates* c. Activation of a context (lines 16–18) is done by trying to change the flag c.isActive from false to true and adding the context to the queue activeContexts if the flag changes. In the main loop of the procedure (lines 4–7) the Todo expressions of every context from activeContexts are repeatedly processed, and the context is *deactivated* when its Todo becomes empty. Processing of expressions (lines 8–12) is done similarly as in Algorithm 4.1, except that all inferences are performed with the context-local Closure.

The main purpose of activation and deactivation of contexts is to make sure that, for each context c, the set c.Closure is never accessed by more than one worker at a time. This allows the use of ordinary datastructures for Closure without the need to deal with concurrency issues. To prove that c.Closure is never accessed concurrently by several workers, consider the moment t_n when c was returned by activeContexts.poll() the n-th time. Then there should be at least n corresponding

Algorithm 4.4: Non-thread-safe activation of contexts

```
1 activate(context) :
2     if context.isActive = false then
3         context.isActive ← true;
4         activeContexts.put(context);
```

calls of activeContexts.put(c) before t_n, and thus, at least n calls to activate(c) in which the value of c.isActive has changed from false to true. Hence, the value of c.isActive must have changed from true to false at least $n-1$ times before t_n, and so, there must be at least $n-1$ calls of deactivate(c). Let t'_{n-1} be the moment of the $(n-1)$-th call of deactivate(c). Then $t_{n-1} < t'_{n-1} < t_n$ since deactivate(c) is only called after c is returned by activeContexts.poll() in the main loop (lines 4–7). Furthermore, for deactivate(c) at time t'_{n-1}, the corresponding activeContexts.poll() returning c must be the one at time t_{n-1}; otherwise, for activeContexts.poll() returning c at time t_{n-1}, the corresponding deactivate(c) happens after t_n, so there are fewer calls to activeContexts.poll() than to deactivate(c) before t_n, which is not possible. Hence c.Closure can be accessed only during the intervals $[t_n; t'_n]$ and only by one worker in each interval. Therefore, two workers can never access c.Closure at the same time.

It is essential that, in the function activate(context), the flag context.isActive is compared and changed in one instruction. The procedure is no longer thread-safe if we replace this function with a seemingly equivalent one in Algorithm 4.4. Indeed, it may well be the case that two workers call activate(c) for some c at the same time, 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 in Algorithm 4.3: if several workers call isActive.compareAndSet(**false**, **true**) at the same time, the operation succeeds only for one of them.

Note that the implementation of the queues 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 context.enqueue(expression). Fortunately, concurrent queues are common in programming libraries.

Next, we show correctness of Algorithm 4.3. To show termination, note that a context c can be inserted into activeContexts only after an expression has been inserted into c.Todo, which happens only for expressions in Input, or after a new expression has been inserted into some Closure. So, if

the number of expressions derived from Input is finite, eventually, the procedure is not going to add anything into activeContexts or any Todo. Therefore, the main loop of the procedure (lines 4–7) terminates.

It is possible to show that, after termination, all Todo queues will be empty. Indeed, consider any context c and the value of c.isActive during the run of the procedure. If it was never true, then nothing has been inserted into c.Todo. Otherwise, let t be the last moment when c.isActive was set to false (at line 20). Then c.Todo is empty after t, because, otherwise, the value of c.isActive would have changed by the subsequent calls of `activate`(c) and `deactivate`(c). Note that checking Todo on emptiness in line 21 is important for correctness. Otherwise, it could happen, e.g., that some worker inserts an expression into c.Todo shortly before the flag c.isActive was set to false in `deactivate(`c`)`, in which case the context c will not be activated.

Finally, it is easy to show by induction that every expression derivable from Input will be inserted into c.Closure for every context c assigned to this expression. This is a consequence of our requirement on the function `getContexts`(expression), which ensures that every inference will be performed within at least one context.

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

In this section we briefly discuss how to turn the saturation procedure for $\mathcal{EL}^+_\bot$ presented in Section 4.2.3 into a concurrent one using Algorithm 4.3. First, we need to find a suitable assignment of contexts to expressions that satisfies the requirement for the inference rules in Fig. 3.1. 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. Unfortunately, this strategy provides only for as many contexts as there are inference rules, so it may not be very effective with many workers. To find a better solution, note that all premises of the rules in Fig. 3.1 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 the following assignment: expressions of the form $\mathsf{init}(C)$, $C \sqsubseteq D$, or $E \xrightarrow{R} C$ are assigned to the context C. Additionally, $E \xrightarrow{R} C$ is also assigned to the context E because the expression may be used as the second premise of rule $\mathbf{R}_\circ$. Note that this assignment of contexts is minimal in the sense that every inference is possible in exactly one

context, so the overall number of inferences performed by the workers does not increase.

Note that since the contexts c under our assignment are parts of expressions, it is possible to 'compress' the representations of elements in c.Todo and c.Closure by removing the arguments that are uniquely determined the context c as below:

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

The context-local version of function `process`(expression) from Algorithm 4.2 that uses this context assignment and representation is presented in Algorithm 4.5.

Example 4.5. In this example we show how the inferences from Example 3.1 are transformed when applying the concurrent saturation procedure in Algorithm 4.3. In Table 4.1 we have listed all conclusions (3.5)–(3.19) in their Todo representation in the order they are derived in the respective contexts. The rules applied to these conclusions by Algorithm 4.5 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 4.2 and 4.4 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 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.

Algorithm 4.5: C.process(expression)

```
1  C.process(Init) :
2      if C.Init = false then
3          C.Init ←true;
4          C.enqueue(new Sub(C));                                   // rule R_0
5          if top.negOccurs > 0 then
6              C.enqueue(new Sub(top));                             // rule R_⊤
7  C.process(Sub(D)) :
8      if C.subs.add(D) then
9          if D = bottom then
10             for each E, R with C.backLinks(E, R) do
11                 E.enqueue(add(new Sub(bottom)));                 // rule R_⊥
12         if D instanceOf IdxConjunction then
13             C.enqueue(new Sub(D.firstConj));
14             C.enqueue(new Sub(D.secondConj));                    // rule R⁻_⊓
15         for each D_2, E with C.subs(D_2) and negConjs(D, D_2, E) do
16             C.enqueue(new Sub(E));                               // rule R⁺_⊓
17         for each D_1, E with C.subs(D_1) and negConjs(D_1, D, E) do
18             C.enqueue(new Sub(E));                               // rule R⁺_⊓
19         if D instanceOf IdxExistential then
20             D.filler.enqueue(Init);
21             D.filler.enqueue(new BackLink(D.role, C));
22             C.enqueue(new ForwLink(D.role, D.filler)) ;          // rule R⁻_∃
23         for each E, F, R, S with C.backLinks(R, E) and negExists(S, D, F) and hier(R, S) do
24             E.enqueue(new Sub(F));                               // rule R⁺_∃
25         for each E with concIncs(D, E) do
26             C.enqueue(new Sub(E));                               // rule R_⊑
27 C.process(BackLink(R, E)) :
28     if C.backLinks.add(⟨R, E⟩) then
29         if C.subs.contains(bottom) then
30             E.enqueue(new Sub(bottom));                          // rule R_⊥
31         for each D, F, S with C.subs(D) and negExists(S, D, F) and hier(R, S) do
32             E.enqueue(new Sub(F));                               // rule R⁺_∃
33         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
34             D.enqueue(new BackLink(S, E));
35             E.enqueue(new ForwLink(S, D));                       // rule R_∘
36 C.process(ForwLink(R, D)) :
37     if C.forwLinks.add(⟨R, D⟩) then
38         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
39             D.enqueue(new BackLink(S, E));
40             E.enqueue(new ForwLink(S, D));                       // rule R_∘
```

Context A :	Context $C \sqcap D$:	Context D :	context : Rule[Precomputed Table Entries]
Init			$A : \mathbf{R_0}$
Sub(A)			$A : \mathbf{R}_{\sqsubseteq}$[concIncs($A, \exists R.(C \sqcap D)$)]
Sub($\exists R.(C \sqcap D)$)			$A : \mathbf{R}^{-}_{\exists}$
ForwLink($R, C \sqcap D$)	Init		$C \sqcap D : \mathbf{R_0}$
	BackLink(R, A)		
	Sub($C \sqcap D$)		$C \sqcap D : \mathbf{R}^{-}_{\sqcap}$
	Sub(C)		
	Sub(D)		$C \sqcap D : \mathbf{R}^{+}_{\exists}$[negExists($S, D, \exists S.D$), hier($R$,$S$)]
Sub($\exists S.D$)			$A : \mathbf{R}^{-}_{\exists}$, $\mathbf{R}^{+}_{\sqcap}$[negConjs($A, \exists S.D, A \sqcap \exists S.D$)], $\mathbf{R}_{\sqsubseteq}$[concIncs($\exists S.D, C$)]
ForwLink(S, D)		Init	$D : \mathbf{R_0}$
Sub($A \sqcap \exists S.D$)		BackLink(S, A)	$A : \mathbf{R}^{-}_{\sqcap}, \mathbf{R}_{\sqsubseteq}$[concIncs($A \sqcap \exists S.D, B$)]
Sub(C)		Sub(D)	$D : \mathbf{R}^{+}_{\exists}$[negExists($S, D, \exists S.D$), hier($S$,$S$)]
Sub(B)			$A : \mathbf{R}_{\sqsubseteq}$[concIncs($B, A \sqcap \exists S.D$)]

Table 4.1: The rule applications from Example 3.1 using the concurrent saturation procedure

Show me a completely smooth operation and I'll show you someone who's covering mistakes. Real boats rock.

—Frank Herbert
Chapterhouse: Dune

Chapter 5

Optimization Techniques

In the previous chapter we showed how one can accelerate the saturation procedure by taking advantage of multiprocessor systems to perform computations in parallel. Another way to make the procedure more efficient is to reduce the number of computations. In this chapter we describe several optimizations of this kind that turned out to work well in practice.

In Sections 5.1 and 5.2 we identify certain redundant inferences which can be omitted without losing completeness, in Section 5.3 we improve treatment of large disjointness axioms, in Section 5.4 we describe join optimizations for matching premises of inference rules, and finally in Section 5.5 we consider the algorithm for constructing the (transitively reduced) taxonomy from the result of the (transitively closed) saturation.

5.1 Optimization of Decomposition Rules

In this section we show that it is possible in some situations to omit applications of the decomposition rules $\mathbf{R}^-_\sqcap$ and $\mathbf{R}^-_\exists$ from Fig. 3.1 without losing completeness. Specifically, we prove the correctness of the following two optimizations:

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

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

Optimization $O_\sqcap$ is trivial: if $C \sqsubseteq D_1 \sqcap D_2$ has been derived by rule $\mathbf{R}^+_\sqcap$, then both the premises $C \sqsubseteq D_1$ and $C \sqsubseteq D_2$ of the rule must have been derived earlier, so it is redundant to derive them

again in rule $\mathbf{R}^-_\sqcap$. Even then, avoiding the actual application of the rule can still bring some speedup of the algorithm.

Optimization $O_\exists$ is more interesting: if $C \sqsubseteq \exists S.E$ has been derived by rule $\mathbf{R}^+_\exists$, then some premises $C \xrightarrow{R} D$ with $R \sqsubseteq^*_\mathcal{O} S$ and $D \sqsubseteq E$ of the rule must have been derived earlier (and thus also $\mathsf{init}(D)$), but not necessarily the conclusion $C \xrightarrow{S} E$ of rule $\mathbf{R}^-_\exists$ applied to $C \sqsubseteq \exists S.E$. This observation motivates the following definition:

Definition 5.1 (Redundancy of $\mathbf{R}^-_\exists$)**.** We say that the inference from $C \sqsubseteq \exists S.E$ by $\mathbf{R}^-_\exists$ is *redundant w.r.t.* Closure *and* $\mathcal{O}$ if $\{\mathsf{init}(D), C \xrightarrow{R} D, D \sqsubseteq E\} \subseteq$ Closure for some concept D and role $R \sqsubseteq^*_\mathcal{O} S$. We say that Closure is *closed under* $\mathbf{R}^-_\exists$ *up to redundancy* w.r.t. $\mathcal{O}$ if every inference using $\mathbf{R}^-_\exists$ from Closure is redundant w.r.t. Closure and $\mathcal{O}$.

Note that if Closure is closed under $\mathbf{R}^-_\exists$ and $\mathbf{R_0}$, then it is closed under $\mathbf{R}^-_\exists$ up to redundancy because $C \sqsubseteq \exists S.E \in$ Closure implies $\{\mathsf{init}(D), C \xrightarrow{S} D, D \sqsubseteq D\} \subseteq$ Closure. So, closure under $\mathbf{R}^-_\exists$ up to redundancy is a weaker condition than closure under $\mathbf{R}^-_\exists$ and $\mathbf{R_0}$. Also note that an inference from $C \sqsubseteq \exists S.E$ by $\mathbf{R}^-_\exists$ can be redundant even if $C \sqsubseteq \exists S.E$ was not obtained by rule $\mathbf{R}^+_\exists$. Therefore, this notion of redundancy can potentially be used to prune even more inferences than $O_\exists$.

The correctness of the optimization $O_\exists$ now follows from Lemma 5.2:

Lemma 5.2. *All results in Section 3.2 hold for any set* Closure *that is closed under* $\mathbf{R}^-_\exists$ *up to redundancy and closed under the remaining inference rules.*

Proof. The condition that Closure is closed under $\mathbf{R}^-_\exists$ was used only in the case $D = \exists S.D_2$ of Lemma 3.5 in . We can repeat this case of the proof using a weaker assumption that Closure is closed under $\mathbf{R}^-_\exists$ up to redundancy:

- Case $D = \exists S.D_2$: Assume that $x_C \in \Delta^\mathcal{I}$ and $C \sqsubseteq D = C \sqsubseteq \exists S.D_2 \in$ Closure. We need to prove that $x_C \in D^\mathcal{I}$, where $\mathcal{I}$ is the canonical model constructed for Closure. Since Closure is closed under $\mathbf{R}^-_\exists$ up to redundancy, we have $\{\mathsf{init}(D_1), C \xrightarrow{R} D_1, D_1 \sqsubseteq D_2\} \subseteq$ Closure for some D_1 and some $R \sqsubseteq^*_\mathcal{O} S$. Since $x_C \in \Delta^\mathcal{I}$, we have $C \sqsubseteq \bot \notin$ Closure, and since $C \xrightarrow{R} D_1 \in$ Closure, due to rule $\mathbf{R}_\bot$, we have $D_1 \sqsubseteq \bot \notin$ Closure. Therefore, $x_{D_1} \in \Delta^\mathcal{I}$ since $\mathsf{init}(D_1) \in$ Closure. By the induction hypothesis applied to $x_{D_1} \in \Delta^\mathcal{I}$ and $D_1 \sqsubseteq D_2 \in$ Closure, we get $x_{D_1} \in D_2^\mathcal{I}$. Since $C \xrightarrow{R} D_1 \in$ Closure and $R \sqsubseteq^*_\mathcal{O} S$, we have $\langle x_C, x_{D_1} \rangle \in S^\mathcal{I}$ by the definition of $S^\mathcal{I}$. Then $x_C \in (\exists S.D_2)^\mathcal{I}$ by the semantics of $\exists$. □

Example 5.3. Using optimization $O_\exists$, the application of rule $\mathbf{R}^-_\exists$ to axiom (3.13) in Example 3.1 is redundant. This avoids deriving expressions (3.15), (3.16), and (3.19), and, in particular, even avoids the need to initialize D.

In practice, to implement $O_\sqcap$ and $O_\exists$ in the saturation algorithm, we introduce a new expression constructor $\mathsf{Sub}^+(C, D)$ and change the implementation of rules $\mathbf{R}^+_\sqcap$ and $\mathbf{R}^+_\exists$ in Algorithm 4.2 to produce Sub^+ instead of Sub. Processing of $\mathsf{Sub}^+(C, D)$ is analogous to `process`($\mathsf{Sub}(C, D)$) in Algorithm 4.2 except that the applications of rules $\mathbf{R}^-_\sqcap$ and $\mathbf{R}^-_\exists$ are omitted. The changes in Algorithm 4.5 are analogous.

It is currently not very clear how one can benefit from the more general optimization based on the notion of redundancy in Definition 5.1. While it is certainly possible to add the redundancy conditions as additional negative premises for rule $\mathbf{R}^-_\exists$ in Fig. 4.2, it is not clear how to implement such a rule efficiently so that it could give further practical improvement over the optimization $O_\exists$.

5.2 Optimization of the Role Composition Rule

All the inference rules in Fig. 3.1 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 $\mathbf{R}_\circ$ 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 $\mathbf{R}_\circ$ from the same premises.

First, to simplify the application of rule $\mathbf{R}_\circ$, 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 $\mathbf{R}_\circ$ in Fig. 4.2 then simplifies to the following:

$\mathbf{R}_\circ$: 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 $\mathbf{R}_\circ$ 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 ... 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 4.2 but the tuple is already there, we know that $C \sqsubseteq C_i$ must have already been derived for at least two different is, so we derive the conclusion $C \sqsubseteq \bot$.

5.4 Efficient Join Computation

Iterations over joins of tables are used extensively in Algorithm 4.2 and Algorithm 4.5 to retrieve all matching side conditions and premises of inference rules. We can optimize such iterations by iterating over smaller tables and precomputing partial joins. In this section we illustrate these techniques on the application of rules $\mathbf{R}^+_\sqcap$ and $\mathbf{R}^+_\exists$.

Consider rule $\mathbf{R}^+_\sqcap$ implemented in Algorithm 4.2 in lines 14–17. To apply the rule with an axiom $C \sqsubseteq D$ as the first premise, the loop in line 14 iterates over all indexed concepts D_2 and E such that $\mathsf{subs}(C, D_2) \wedge \mathsf{negConjs}(D, D_2, E)$, 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 with $\mathsf{subs}(C, D_2)$ 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 pairs of D_2 and E with $\mathsf{negConjs}(D, D_2, E)$ and for each of them check if $\mathsf{subs}(C, D_2)$ holds. Most concepts D occur only in few negative conjunctions, or even in none at all; for such concepts the second order of iteration will be more efficient than the first. In some ontologies, however, there are several concepts that occur in many conjunctions. For example, in the SNOMED CT ontology there are three concepts that occurs in more than 1,000 negative conjunctions. For such concepts the first order of iteration is likely to be more efficient than the second. For this reason, we do not fix the order of iteration upfront, instead, we dynamically choose the order that requires iterating over a smaller set.

The above approach can be further improved for rules where the same join is recomputed multiple times. Consider the application of rule $\mathbf{R}^+_\exists$ to a link $E \xrightarrow{R} C$ as the first premise implemented in Algorithm 4.2 in lines 29–30. For simplicity, we will ignore the role hierarchy in this discussion. In that case, the loop in line 29 iterates over all indexed concepts D and F such that $\mathsf{subs}(C, D) \wedge \mathsf{negExists}(R, D, F)$, 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 all links $E \xrightarrow{R} C$ with the same R and C. To avoid the recomputation, we maintain the join in a new table $\mathsf{props}(R, C, F) := \mathsf{subs}(C, D) \bowtie \mathsf{negExists}(R, D, F)$ which we update whenever a new subsumption $C \sqsubseteq D$ is derived.

5.5 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 subsump-

tion 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*. The problem of efficient taxonomy construction has been investigated in the literature both in the context of tableau-based (see, e.g., [39]) and saturation-based (see, e.g., [16]) reasoning.

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

Algorithm 5.1: Naive Transitive Reduction

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

Algorithm 5.2: Better 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 ∈ 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);
```

A naive algorithm for computing A.directSubs is shown in Algorithm 5.1. 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 5.2. Note that it is safe to execute Algorithm 5.2 in parallel for multiple concepts A.

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 5.4. Consider again the ontology from Example 3.1. Initializing the saturation algorithm with $\mathsf{init}(A)$ for each atomic concept A occurring in the ontology and projecting the computed subsumptions in table subs to atomic concepts yields

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

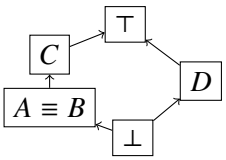
from which Algorithm 5.2 computes the following taxonomy (shown to the right):

$A.\mathsf{equivalent} = \{A, B\}$, $A.\mathsf{directSubs} = \{C\}$,

$B.\mathsf{equivalent} = \{A, B\}$, $B.\mathsf{directSubs} = \{C\}$,

$C.\mathsf{equivalent} = \{C\}$, $C.\mathsf{directSubs} = \emptyset$,

$D.\mathsf{equivalent} = \{D\}$, $D.\mathsf{directSubs} = \emptyset$.



To know a thing well, know its limits. Only when pushed beyond its tolerances will its true nature be seen.

—Frank Herbert
Chapterhouse: Dune

Chapter 6

Experimental Evaluation

A practical implementation of the reasoning methods explained in the previous chapters is provided in the form of the ontology reasoner ELK. This chapter describes the ELK system (Section 6.1) and provides an experimental evaluation of classification using ELK on some of the largest $\mathcal{EL}^+_\bot$ ontologies (described in Section 6.2) that we were able to obtain from public and commercial sources.

In Section 6.3 we compare the performance of ELK in its default settings (all optimizations turned on) with other commonly used OWL and OWL EL reasoners. In Section 6.4, to evaluate the effect of individual optimizations, we repeat the experiments with certain optimizations turned off. In Section 6.5 we evaluate the effect of varying the number of concurrent workers. Finally, in Section 6.6 we compare the two transitive reduction algorithms from Section 5.5.

6.1 System Overview

ELK is a Java-based system for reasoning with OWL EL ontologies under Direct Semantics; as explained in Section 2.3, OWL EL can be viewed as a slightly richer syntax for the $\mathcal{EL}$ family of DLs. ELK is free and open source, using a commercial-friendly Apache 2 license.[1]

The latest stable release ELK 0.3.1 supports all features of $\mathcal{EL}^+_\bot$: 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

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

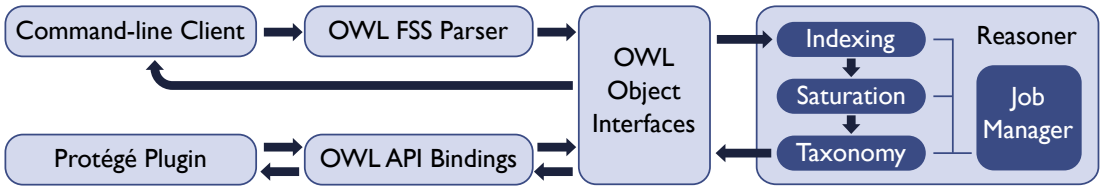


Figure 6.1: Main software modules of ELK and information flow during classification

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 procedure described in Chapter 4 and all optimization techniques detailed in Chapter 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 , saturation of concepts, 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 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. 6.1. 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 uses ELK's bindings to the OWL API [55] to get this data from Protégé [66]. All further processing is based on ELK's own representation 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 has been descibed in detail in the previous chapters.

Useful combinations of ELK's modules are distributed in three pre-built packages, each of

which includes the ELK reasoner. 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*.[2] The *OWL API bindings* package allows ELK to be used as a software library that is controlled via the OWL API interfaces.

6.2 Experimental Setup

The experiments in this chapter 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` and 4GB of heap space. All figures reported here 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 OWL functional syntax using the supplied converter. Additionally, we used an 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.[3] We included several versions of GALEN. GALEN7 and GALEN8 were obtained from versions 7 and 8 respectively of OpenGALEN[4] 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 OWL 2 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[5] by removing inverse role and functional role axioms. This version of the ontology has been used extensively in the past for evaluating reasoners [31, 60, 63, 88, 116]. It is similar to GALEN7 but its only role compositions are transitivity axioms. We also used two versions of the Gene Ontology[6] which we call GO1 and GO2. The older GO1, published in 2006,

[2] `http://www.b2international.com/portal/snow-owl`
[3] `http://www.ihtsdo.org/licensing/`
[4] `http://www.opengalen.org/sources/sources.html`
[5] `http://www.co-ode.org/galen/`
[6] `http://www.geneontology.org`

	$C \sqsubseteq D$	$C \equiv D$	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 6.1: Ontology metrics: number of axioms

	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

Table 6.2: Ontology metrics: number of concepts, roles, and constructors by occurrence polarities

has been used in many performance experiments [15, 31, 39, 60, 88, 116, 133]. 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 [122] and the Ontobee [141] 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.[7]

[7] `http://code.google.com/p/elk-reasoner/wiki/TestOntologies`

Tables 6.1 and 6.2 show various statistics about the ontologies from our benchmark suite. Table 6.1 shows the number of various axiom types; the only logical axiom not mentioned in the table is one reflexive role axiom in ANATOMY. Table 6.2 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 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 6.1 and 6.2. 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 $\mathbf{R}^{+}_{\exists}$ to the positive existentials in these ontologies even though, due to lack of negative existentials, the resulting links can never participate in rule $\mathbf{R}^{-}_{\exists}$.

All of our experiments are focused on terminological reasoning, which is currently the most common reasoning problem used in applications involving $\mathcal{EL}$ ontologies [38, 50, 58, 99, 103]. 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 [30], Grail [104], or frame-like languages, which did not have those features. In our previous experiments with nominals [65], 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.

6.3 Performance Comparison with Other Reasoners

We compared ELK 0.3.1 to the specialized OWL EL reasoners CEL 1.1.2 [15], jcel 0.18.0 [87], the REL reasoner from TrOWL 1.2 [130], and Snorocket 2.0.5 [78], to general OWL reasoners

FaCT++ 1.6.0 [132], HermiT 1.3.6 [94], JFact 0.9,[8] Pellet 2.3.0 [121], and RacerPro 2.0 build 20121209 [47], and to experimental consequence-based reasoners CB r.12 [60] and ConDOR r.12 [120]. We ran all reasoners in their default settings.

CEL, jcel, REL, and Snorocket are typical OWL EL systems that implement completion-based procedures [11]. 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 7.1 provides further details, and also discusses various other systems that we have not included in this evaluation. Recent versions of Snorocket implement similar context-based concurrency techniques as we described in Section 4.4; 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, 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 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 6.3 shows the wall-clock time each reasoner spent executing the OWL API 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 6.4 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

[8] `http://jfact.sourceforge.net/`

	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 6.3: 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

Table 6.4: Loading + classification time in seconds, measured using the OWL API

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 6.5 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 $\mathbf{R}_\circ$ 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

	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

Table 6.5: Classification time in seconds as reported by the reasoner

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 6.3. 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.

6.4 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 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 ran the classification algorithm on each complex ontology in five configurations: with none of the three optimizations $O_{\sqcap}$, $O_{\exists}$, and $O_{\circ}$ (which corresponds to applying all the inference rules to all applicable premises), with one of these optimizations turned on at a time, and with all the three optimizations together (which is 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 6.6.

First, we observed that on SNOMED CT no link $C \xrightarrow{R} D$ is derived more than once. This is because, even though SNOMED CT contains one role composition axiom, the role composition

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

Table 6.6: Classification time in seconds (1 working thread) and number of derived axioms

rule $R_\circ$ is never applied on this ontology. The links are therefore derived only by rule $\mathbf{R}^-_\exists$ 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 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 $\mathbf{R}^+_\sqcap$ 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 $C \xrightarrow{R} D$ 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 6.6; 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 6.6 on all the test ontologies apart from GO2, with a reduction in classification time by 25% on SNOMED CT, by 43% on ANATOMY, by 20% on GALEN-OWL, by 32% on GALEN7, and by 50% on GALEN8.

The optimization $O_\circ$ avoids the derivation of some links by rule $\mathbf{R}_\circ$ 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 $\mathbf{R}^+_\exists$, 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 $\mathbf{R}_\circ$ 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 it reduces the classification time by

24%. Unlike the previous two optimizations, $\mathbf{R}_\circ$ 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 reducing the classification time by 29% on SNOMED CT and by as much as 60% 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.

6.5 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 6.7.[9]

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 6.7 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 clas-

[9]For a fair comparison with other reasoners, we ran ELK in the experiments in Section 6.3 through the 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 6.7 and those in Table 6.3.

		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

Table 6.7: Classification time in seconds and relative speedup for increasing number of concurrent workers

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

6.6 Transitive Reduction

Finally, we evaluated the difference between the 'naive' Algorithm 5.1 and the 'better' Algorithm 5.2 for transitive reduction from Section 5.5. For this experiment, we implemented the two algorithms exactly as shown in Section 5.5 even though the naive algorithm is incorrect in the presence of equivalent concepts. For each of the two algorithms, Table 6.8 shows the running time and the number of passes through the inner for loop of the algorithm.

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

Table 6.8: Running time in seconds and the number of passes through the inner loop of the two transitive reduction algorithms from Section 5.5

The experiments show that the better algorithm is about 2–3 times faster than the naive algorithm on SNOMED CT, ANATOMY, GALEN7, and GALEN8. The better algorithm always requires substantially fewer passes through the inner loop, with the exception of the EMAP ontology, which entails no 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 'better' algorithm also performs set removals, which are more expensive than membership checks performed by the 'naive' algorithm.

There is no real ending. It's just the place where you stop the story.

—Frank Herbert

Chapter 7

Discussion

In this chapter we discuss related work for three different aspects of our contribution: OWL EL reasoning (Section 7.1), rule- and saturation-based reasoning (Section 7.2), and concurrent and distributed reasoning (Section 7.3). We conclude in Section 7.4.

7.1 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 [8, 13]. Most OWL EL implementations use variations of so-called *completion-based algorithms* first proposed for $\mathcal{ELH}$ [25] and subsequently extended to $\mathcal{EL}^{++}$ [10]. Later works modify this approach to also cover reflexive roles and range restrictions [11], Boolean role constructors [107], and local reflexivity [70]. The difference between completion-based and consequence-based algorithms is subtle and mainly presentational: the first construct completion graphs (canonical models), whereas the second derive consequences of axioms. As we have seen in Section 3.2, one view can be converted into the other. Some consequence-based algorithms, like the one presented here, do not require the input ontology to be normalized as for completion-based algorithms [10, 11, 25]. One can, however, see close similarities between ontology normalization and the indexing phase described in Section 4.2.1. The algorithm presented here is also closely related to the *proof-theoretic* algorithm for $\mathcal{EL}$ [52], which does not require normalization either.

A number of reasoners have been implemented for the $\mathcal{EL}$ family. The first such system, CEL [15], implemented a part of the $\mathcal{EL}^{++}$ classification algorithm [10]. Several optimization techniques

used in CEL, such indexing of side conditions of inference rules [14] and improved taxonomy construction [16], are similar to those used in ELK. Various later systems have reimplemented the 'CEL algorithm'[1] 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 [78], TrOWL REL [130], and jcel [86]. Recent prototype Cheetah was used to investigate the application of linear-time algorithms for propositional Horn logic in $\mathcal{EL}^+$ reasoning [116]. 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 to similar conclusions when experimenting with prototype versions of ELK for reasoning with role chains [64] and (unrestricted) nominals [65]. For example, it is hard to come up with examples that would require non-safe use of nominals in OWL EL ontologies. Supporting safe nominals, as described in Appendix A, 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 [31] and OREL [73] explored the use of relational database 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 [35], using the DReW reasoner [142] to implement a rule-based calculus for OWL EL [70]. 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 [45]. 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++ [132] reduces the number of subsumption tests for *completely defined concepts*, which frequently occur in GO1 and SNOMED CT [133]. An extension of this optimization with *structural pseudo-model embedding* has been successfully used by RacerPro to classify SNOMED CT [89]. HermiT [94] uses an optimization that can completely avoid subsumption tests for *deterministic ontologies* (including $\mathcal{EL}$) [39]: subsumptions can be just read out of the models produced for concept satisfiability tests. The latest version of Pellet [121] can apparently

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

switch to a specialized algorithm when the ontology is within a fragment of OWL EL.[2] HermiT and Pellet, however, were still unable to classify SNOMED CT in our experiments. The CB reasoner uses a consequence-based algorithm for Horn-$\mathcal{SHIF}$ [60], which works similar to the algorithm 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 [87].

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 [75, 68], least common subsumer computation [13, 17], unification [9], and interpolation [97]. These reasoning services have yet to make it into common tools, although prototype implementations exist.

7.2 Rule- and Saturation-Based Reasoning

Inference rules 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 [37].

The abstract saturation procedure described in Section 4.1 is inspired by the *given clause* approach / set of support strategy in theorem proving [139], 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 [37]. Related methods are the linear evaluation strategy for Horn rules studied in Cheetah [116], and ELK's partial join computation described in Section 5.4. These results show that this approach can be useful in OWL reasoning but does not pay off in all cases.

Rule-based approaches have also been extensively applied to reasoning in the OWL 2 Profiles [72]. OWL EL was discussed in Section 7.1 before. Another common use of rule-based calculi is instance retrieval in OWL RL [90] and its sublanguages, especially pD^* (a.k.a. OWL-Horst) [128] and RDFS [26]. 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

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

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 [45] and OWL RL [115, 127]. 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 [71].

For further optimizing the application of rules, various works on OWL RL distinguish between static/pre-computed and dynamic/inferred premises of inference rules [54, 135]. 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 [54]. 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 5.4. 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 5.4 is a simple form of join order optimization.

7.3 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 [21], tableau algorithms that explore non-deterministic choices concurrently [23, 80, 85, 140], a resolution calculus for $\mathcal{ALCHIQ}$ where inferences are exchanged between distributed workers [109], and a distributed classification algorithm that can be used to concurrently invoke (serial) OWL reasoners for checking relevant subsumptions [6, 7]. 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 $\mathcal{EL}^+$ [95]. However, this idea has not been empirically evaluated, and it has been argued that it ignores several practical problems [110]. Saturation-based reasoning with shared memory has recently been explored for RDFS [49]. This approach also investigates the use of alternative computation platforms, such as many-core GPUs, which bears some challenges related to memory management.

Many other works focus on *distributed* reasoning using many machines, instead of shared-memory parallelism using 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 [96, 124]. Another approach to partitioning in OWL is the computation of *modules* [28], which has also been considered for distribution and related optimizations recently [4, 134].

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 [53, 69, 136, 137]. 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 [53, 54, 95, 135, 136, 137] and indexing of rules [54] 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, 139], and it may seem that concurrent extensions of this procedure as described in Section 4.3 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 [81], 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.

7.4 Conclusions and Future Work

We have described the main theoretical, algorithmic, and implementation techniques that make ELK one of the most competitive ontology reasoners available today. Indeed, despite its relatively short history, ELK has already been used in a variety of biomedical applications [50, 51, 99, 129]. Often, ELK is the only reasoner that is able to deal with large volumes of data used in these applications.

From our experiments in Chapter 6, we can summarize that the most significant performance improvement results from the concurrent saturation procedure (Section 4.4), achieving a speedup factor as high as 3.8. This improvement, however, may depend on the number of processors / cores available. The second most useful improvement comes from the optimization of inference rules, in particular, avoiding decomposition of negative existential restrictions (Section 5.1). The speedup factor here was reaching 2.5 (corresponding to 60% reduction). The combinations of these improvements can result in more than 8 times speedup, such as in the case of GALEN8. It is difficult to estimate how much of the good performance of ELK is due to other techniques, such as indexing (Section 4.2.1) and efficient joint computation (Section 5.4), since those features cannot be easily switched off.

In this presentation we have focused only 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 [45]. 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 goal-directed way, such as computation of direct subsumers for concepts. While not necessarily improving performance, these features may certainly widen possible uses of ELK. Since ELK is currently under heavy development, 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.

There are many interesting directions for future work. Not all OWL EL features are currently covered by ELK. We have studied ‘pay-as-you-go’ extensions of our approach to nominals [65],

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 [83]. This result can be used even with certain datatype restrictions outside of the OWL EL profile. The notion of context introduced in Section 4.3 provides a natural way to localizing inferences. Apart from performing inferences in parallel, this has been recently used for implementing incremental reasoning in ELK [62], and can also be potentially useful for axiom pinpointing and debugging.

Tractable ontology reasoning algorithms are only a first step towards obtaining efficient reasoning systems. Careful design, optimization, and implementation, play an equally important role. 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.

Part III

Beyond Horn DLs

Major flaws in government arise from a fear of making radical internal changes even though a need is clearly seen.

—Frank Herbert
Chapterhouse: Dune

Chapter 8

Consequence-Based Reasoning in $\mathcal{ALCI}$

In this chapter we introduce a general consequence-based framework for subsumption reasoning in the DL $\mathcal{ALCI}$. Moreover, via well-known reductions of transitivity and role inclusion axioms, the framework can also be applied in the DL $\mathcal{SHI}$. The framework can be used to solve the following reasoning task: given an ontology $\mathcal{O}$ and a finite set of subsumption queries $\mathcal{Q}$, for each query $q \in \mathcal{Q}$ determine whether $\mathcal{O} \models q$. Note that this captures ontology classification, which involves queries of the form $A \sqsubseteq B$ for A and B atomic concepts from $\mathcal{O}$.

For convenience, unlike in the previous chapters, here we require that the input ontology $\mathcal{O}$ is normalized as described in Section 8.1 below. In Section 8.2 we discuss the intuitions behind our framework, and in Section 8.3 we introduce the framework formally. In Section 8.4 we discuss redundancy elimination techniques that can be used to optimize reasoning. In Section 8.5 we discuss several concrete instantiations of our framework. Finally, in Section 8.6 we present first experimental results.

8.1 Normal Form

In this section we present a suitable form of normalized axioms, inspired by clauses in first-order logic, which we use throughout the rest of this thesis. A concrete normalization method based on structural transformation can be found in Appendix B.

A *literal* is a concept of the form A, $\exists R.A$, or $\forall R.A$, for A an atomic concept and R a (possibly inverse) role. The set of all literals is denoted Σ_L. We reserve the letter L for literals, K for con-

junctions of literals, and M for disjunctions of literals. We identify conjunctions and disjunctions of literals with sets of literals (i.e., conjunctions and disjunctions of literals are unordered and without repeated literals) and we use them in standard set operations; furthermore, we identify the empty conjunction and the empty disjunction with $\top$ and $\bot$, respectively.

A *clause* is a concept inclusion of the form $\bigsqcap_{i=1}^{m} L_i \sqsubseteq \bigsqcup_{i=m+1}^{n} L_i$ where $0 \leq m \leq n$ and each L_i is a literal. The clause is *normal* if each L_i with $1 \leq i \leq m$ is an atomic concept. The clause is a *query* if each L_i with $m+1 \leq i \leq n$ is an atomic concept. Conjunction K in a clause $K \sqsubseteq M$ is the *antecedent*, and disjunction M is the *consequent*. A clause $K \sqsubseteq M$ is *over* a set of literals $\mathbf{L}$ if $K \cup M \subseteq \mathbf{L}$. A clause $K' \sqsubseteq M'$ is a *strengthening* of a clause $K \sqsubseteq M$ if $K' \subseteq K$ and $M' \subseteq M$; furthermore, $K \sqsubseteq M \mathrel{\hat{\in}} \mathcal{N}$ means that a set of clauses $\mathcal{N}$ contains at least one strengthening of $K \sqsubseteq M$.

A $\mathcal{SHI}$ ontology $\mathcal{O}$ is *normalized* if each concept inclusion in $\mathcal{O}$ is a normal clause. Note that a normalized $\mathcal{ALCI}$ ontology is simply a set of normal clauses, whereas a normalized $\mathcal{SHI}$ ontology can additionally contain transitivity and role inclusion axioms. In Appendix B we show that, by applying structural transformation and elimination of transitivity and role inclusion axioms, every $\mathcal{SHI}$ ontology $\mathcal{O}$ can be transformed in polynomial time into a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}'$ such that $\mathcal{O}'$ entails the same consequences as $\mathcal{O}$ over the atomic concepts occurring in $\mathcal{O}$. The algorithms presented in the rest of this thesis take a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$ and a finite set of queries $\mathcal{Q}$, and they compute the status of $\mathcal{O} \models q$ for each query $q \in \mathcal{Q}$.

We observe that, when applied to an $\mathcal{EL}$ ontology, our normalization produces only clauses of the form $\bigsqcap_i A_i \sqsubseteq B$, $A \sqsubseteq \exists T.B$, and $A \sqsubseteq \forall T^{-}.B$, and when applied to a DL-Lite$_{horn}$ ontology, it produces only clauses of the form $\bigsqcap_i A_i \sqsubseteq B$, $A \sqsubseteq \bot$, $A \sqsubseteq \exists R.C_{\top}$, and $\top \sqsubseteq \forall R.B$. Here $A_{(i)}$ and B are atomic concepts, T is an atomic role, R is a (possibly inverse) role, and $C_{\top}$ is a distinguished atomic concept produced by the structural transformation of the top concept. Note that all of these clauses are *Horn*—that is, their consequents contain at most one literal.

8.2 Intuitions

In this section we fix the normalized ontology $\mathcal{O}$ and the query q as specified in Example 8.1 below, and then we show how one can prove that $\mathcal{O} \models q$ holds using our framework.

Example 8.1. Let $\mathcal{O}$ be the ontology consisting of clauses (8.1)–(8.10), and let $q = A \sqsubseteq G$; one can

readily verify that $\mathcal{O} \models q$.

$$A \sqsubseteq A_i \quad \text{for } 1 \leq i \leq n \tag{8.1}$$

$$A_i \sqsubseteq C_i \quad \text{for } 1 \leq i \leq n \tag{8.2}$$

$$B \sqsubseteq B_i \quad \text{for } 1 \leq i \leq n \tag{8.3}$$

$$B_i \sqsubseteq C_i \quad \text{for } 1 \leq i \leq n \tag{8.4}$$

$$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq C \tag{8.5}$$

$$C \sqsubseteq \exists R.B \tag{8.6}$$

$$C \sqsubseteq \forall R.D \tag{8.7}$$

$$B \sqcap D \sqsubseteq E \sqcup \forall R^{-}.G \tag{8.8}$$

$$E \sqsubseteq \forall R^{-}.F \tag{8.9}$$

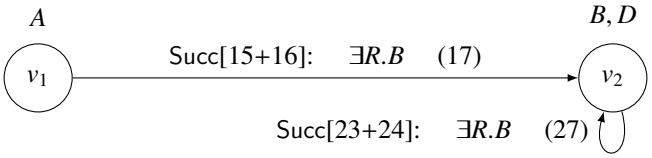
$$A \sqcap F \sqsubseteq G \tag{8.10}$$

Resolution decision procedures [36] can decide whether $\mathcal{O} \models A \sqsubseteq G$ holds by transforming $\mathcal{O}$ into a set of first-order clauses, adding clauses $A(a)$ and $\neg G(a)$ obtained from the negation of the theorem $A \sqsubseteq G$ that is to be proved, and then saturating the result using a suitable first-order resolution variant [18]. Such algorithms are typically worst-case optimal, and they can solve many practically-relevant problems; however, on complex ontologies they can easily run into a combinatorial explosion [93]. In our example, from clauses (8.2)–(8.5) resolution can derive 3^n clauses of the form $L_1 \sqcap \ldots \sqcap L_n \sqsubseteq C$ with $L_i \in \{A_i, B_i, C_i\}$, thus covering all possible combinations of atomic concepts that might be relevant. In contrast, when applied to $\mathcal{O}$, the hypertableau algorithm[1] does not run into this problem: the algorithm is initialized using the fact $A(a)$, and then it derives $A_i(a)$ and $C_i(a)$ for each $1 \leq i \leq n$ and $C(a)$; thus, the algorithm does not consider the "irrelevant" combinations of A_i, B_i, and C_i.

The hypertableau algorithm can thus be seen as being more "goal directed" than resolution. However, the hypertableau algorithm can construct a very large tree of individuals most of which are indirectly blocked, and can thus perform a lot of redundant computation since each indirectly-blocked individual is a "copy" of a nonblocked or directly blocked individual. Resolution is not susceptible to such problems: clauses introduced by resolution are universally quantified and are (unlike individuals in the tableau algorithm) not localized to a specific part of a model. Resolution can thus describe all relevant combinations of atomic concepts using exponentially many clauses, thereby avoiding redundant computation.

Our consequence-based algorithm can be understood as a hybrid between resolution and the hypertableau algorithm. It does not explicitly construct a model; instead, it uses resolution to compute clauses that describe a model $\mathcal{I}$ of $\mathcal{O}$ refuting the relevant queries in $\mathcal{Q}$. The inferences of

[1] A variant of the hypertableau algorithm for normalized $\mathcal{ALCI}$ ontologies is described in Appendix C.



Initialization:	$\top \sqsubseteq A$	(11)	Succ[15+16]:	$\top \sqsubseteq B$	(18)
Hyper[8.1+11]:	$\top \sqsubseteq A_i$ for $1 \le i \le n$	(12)	Succ[15+16]:	$\top \sqsubseteq D$	(19)
Hyper[8.2+12]:	$\top \sqsubseteq C_i$ for $1 \le i \le n$	(13)	Hyper[8.3+18]:	$\top \sqsubseteq B_i$ for $1 \le i \le n$	(20)
Hyper[8.5+13]:	$\top \sqsubseteq C$	(14)	Hyper[8.4+20]:	$\top \sqsubseteq C_i$ for $1 \le i \le n$	(21)
Hyper[8.6+14]:	$\top \sqsubseteq \exists R.B$	(15)	Hyper[8.5+21]:	$\top \sqsubseteq C$	(22)
Hyper[8.7+14]:	$\top \sqsubseteq \forall R.D$	(16)	Hyper[8.6+22]:	$\top \sqsubseteq \exists R.B$	(23)
Pred[15+17+26]:	$\top \sqsubseteq F \sqcup G$	(28)	Hyper[8.7+22]:	$\top \sqsubseteq \forall R.D$	(24)
Hyper[8.10+11+28]:	$\top \sqsubseteq G$	(30)	Hyper[8.8+18+19]:	$\top \sqsubseteq E \sqcup \forall R^-.G$	(25)
			Hyper[8.9+25]:	$\top \sqsubseteq \forall R^-.F \sqcup \forall R^-.G$	(26)
			Pred[23+27+26]:	$\top \sqsubseteq F \sqcup G$	(29)

Figure 8.1: Example inferences of the consequence-based algorithm

the algorithm are not localized to a part of $\mathcal{I}$, which allows our algorithm to avoid redundant computation: although there are many technical differences, our algorithm is somewhat related to the tableau algorithms with caching [43, 44]. To be goal oriented, our algorithm constructs a *context structure*—a graph-like structure whose vertices are called *contexts*. Intuitively, each context describes one or more elements in the model $\mathcal{I}$, and the edges between contexts capture the relations between the corresponding elements of $\mathcal{I}$. Each context v is associated with a set core(v) of *core* atomic concepts. Intuitively, the concepts in core(v) hold for each element of $\mathcal{I}$ that corresponds to v; hence, core(v) determines the "type" of v and the corresponding model elements. Furthermore, each context v is associated with a set of clauses $\mathcal{S}(v)$ that, as in propositional resolution, describe the domain elements in $\mathcal{I}$ that correspond to v. Each clause $K \sqsubseteq M \in \mathcal{S}(v)$ is "relative" to core(v) and should be interpreted as core(v) $\sqcap K \sqsubseteq M$. In other words, since the concepts in core(v) hold in all elements corresponding to v, we drop core(v) from the clauses in $\mathcal{S}(v)$ for the sake of clarity.

The derivation rules of our algorithm will be introduced formally in Table 8.1 in Section 8.3. Here we demonstrate how to use these rules to prove $\mathcal{O} \models A \sqsubseteq G$. We will construct the context structure shown in Figure 8.1; each context is shown as a circle, the core of each context is shown above the circle, the clauses belonging to the context are shown below the circle, and the numbers next to the clauses correspond to the order of inference rule applications. To avoid the drawbacks outlined earlier, our algorithm does not perform inferences between the clauses in $\mathcal{O}$; instead, each inference involves either a single set of clauses $\mathcal{S}(v)$ and possibly the ontology $\mathcal{O}$, or a pair of sets

of clauses $\mathcal{S}(v)$ and $\mathcal{S}(u)$. To initiate the inference process, we initialize the algorithm according to the target query. Since our goal is to prove $\mathcal{O} \models A \sqsubseteq G$, we introduce a single context v_1 with $\text{core}(v_1) = \{A\}$, and we add clause (11) to $\mathcal{S}(v_1)$. Intuitively, this says that the model $\mathcal{I}$ must contain at least one element in which A holds, which is similar to initializing the hypertableau algorithm by $A(a)$. We then use the Hyper rule to derive clauses (12)–(16). Since only A is assumed to hold in context v_1, we derive only a linear number of clauses: no B_i holds in v_1, so we do not derive clauses with irrelevant combinations of atomic concepts. These inferences are analogous to the inferences of the Hyp-rule in the hypertableau algorithm, with the difference that the conclusions are not localized: they hold for each element of $\mathcal{I}$ that corresponds to v_1.

Clause (15) says that the elements in $\mathcal{I}$ corresponding to v_1 must have a successor in which B holds. To satisfy this requirement, we use the Succ rule to introduce context v_2 and add the edge (17) from v_1 to v_2; the edge is labeled with the concept $\exists R.B$ that it satisfies. The Succ rule combines the $\exists$-rule and the $\forall^+$-rule from the hypertableau algorithm by means of sets of atomic concepts $\mathbf{B}_k$ and $\mathbf{B}_p$. Set $\mathbf{B}_k$ contains atomic concepts known to hold in v_2 due to universal restrictions—that is, those concepts L for which $\top \sqsubseteq \forall R.L$ has been derived in v_1. In our example, clause (16) ensures that D holds in v_2, and so we have $\mathbf{B}_k = \{D\}$. Set $\{B\} \cup \mathbf{B}_k = \{B, D\}$ thus provides us with an "upper bound" on the core of the new context: both B and D necessarily hold in context v_2, but we can use an arbitrary subset of $\{B, D\}$ as the core of v_2. Choosing smaller cores might reduce the number of contexts, but might also increase the number of clauses per context. The decision which subset to use is determined by a *strategy*, which is supplied as parameter to our algorithm. In this example we use the "eager" strategy that always uses the maximal subset and thus sets $\text{core}(v_2) = \{B, D\}$; however, we discuss other reasonable strategies in Section 8.5. The Succ rule also initializes $\mathcal{S}(v_2)$ with clauses (18) and (19) specifying that B and D hold in each element of $\mathcal{I}$ that is represented by context v_2. Set $\mathbf{B}_p$ contains atomic concepts that can possibly hold in v_2 due to universal restrictions—that is, those concepts L for which $K \sqsubseteq M \sqcup \forall R.L$ has been derived in v_1 for some K and M. Thus, set $\{B\} \cup \mathbf{B}_p$ provides us with an "upper bound" on the concepts that might need to be considered in v_2. Since (16) is the only clause in $\mathcal{S}(v_1)$ containing a universal restriction, in our example we have $\mathbf{B}_p = \mathbf{B}_k = \{D\}$. Sets $\mathbf{B}_k$ and $\mathbf{B}_p$ need not coincide in general, and the Succ rule adds a clause of the form $L \sqsubseteq L$ for each concept L from $\mathbf{B}_p$ that is not in the core of the new context.

We next use the Hyper rule to derive clauses (20)–(26) in $\mathcal{S}(v_2)$. As in the hypertableau algorithm, since only B holds in context v_2, we do not derive a clause involving A_i. Now clause (23) requires an edge labeled with $\exists R.B$; due to clause (24), the core of the target context can be any subset of $\{B, D\}$; and due to the "eager" strategy, the core of the target context will be $\{B, D\}$. But then, there is no need to introduce a fresh context: our "eager" strategy can instruct the Succ rule to reuse the existing context v_2 since we already have $\text{core}(v_2) = \{B, D\}$. Consequently, we introduce edge (27) using the Succ rule. This "reuse" of v_2 is similar to blocking in the hypertableau algorithm, but is actually much more effective in eliminating redundant computations. First, we never need more than exponentially many contexts, whereas the hypertableau algorithm can constructs trees of doubly exponential size. Second, the clauses belonging to each context are not localized to a specific place in $\mathcal{I}$, and so our algorithm draws the inferences for a particular core only once. In contrast, to ensure that the labels of s and s' coincide and thus ensure the blocking condition, the hypertableau algorithm must draw the same conclusions for s and s'; furthermore, an individual can directly block exponentially many other individuals, so the potential for redundant work is even higher.

Clause (26) says that each element in $\mathcal{I}$ corresponding to a predecessor of an element represented by v_2 must satisfy F or G; hence, we apply the Pred rule to edge (17) and clauses (15) and (26) to derive clause (28). Furthermore, we also apply the Pred rule to edge (27) and thus derive clause (29). The Pred rule essentially "pulls" the information from the successor to the predecessor; however, unlike the $\forall^-$-rule in the hypertableau algorithm, it simultaneously deals with several universal restrictions.

Finally, we use the Hyper rule to derive clause (30), at which point no further inferences are possible. Since all clauses are "relative" to the core of the corresponding context, clause (30) actually corresponds to $A \sqsubseteq G$, so we have proved $\mathcal{O} \models A \sqsubseteq G$. In fact, due to (30), we know that $\mathcal{O} \models K \sqsubseteq M$ for each query $K \sqsubseteq M$ such that $A \sqsubseteq G$ is a strengthening of $K \sqsubseteq M$. Our algorithm is thus not just refutationally complete: for each query $K \sqsubseteq M$ such that $\mathcal{O} \models K \sqsubseteq M$, it derives at least one strengthening of $K \sqsubseteq M$ in each context that *covers* (cf. Definition 8.3) the query.

To further refine our algorithm, we use the *ordered* resolution variant [18]: we parameterize our algorithm with an ordering on literals, and, for each clause $K \sqsubseteq M \in \mathcal{S}(v)$, we apply the inference rules only to literals that are maximal in M w.r.t. the ordering. This can, however, compromize

completeness: clause (30) can be derived in context v_1 from (28) only if G is smaller than F in the ordering. Therefore, to guarantee that the clause will indeed be derived in context v_1, we use a different ordering $\prec_v$ per context v and require that G is smallest in $\prec_{v_1}$. Furthermore, the Pred rule introduces a similar complication: clause (26), which is needed for the Pred rule, can be derived from (25) only if $\forall R^-.G$ is smaller than E in $\prec_{v_2}$. Therefore, to guarantee completeness, we require that $\prec_{v_2}$ must be *R-admissible* (cf. Definition 8.2)—that is, each literal of the form $\forall \mathsf{inv}(R).A$ must be smallest in $\prec_{v_2}$.

8.3 Formalization

In this section we formalize the intuitions that we discussed in Section 8.2. We start by defining the notion of a literal ordering, which we will use later to restrict applicability of our inference rules.

Definition 8.2. A *literal ordering* $\prec$ is a strict partial order (i.e., an irreflexive and transitive relation) on the set Σ_L of all literals. A literal $L \in \Sigma_L$ is $\prec$-*minimal* if no literal $L' \in \Sigma_L$ exists such that $L' \prec L$; moreover, a set of literals N is $\prec$-*minimal* if each literal in N is $\prec$-minimal. A literal $L \in \Sigma_L$ is $\prec$-*maximal* w.r.t. a set of literals N, written $L \not\prec N$, if no literal $L' \in N$ exists such that $L \prec L'$. Given a role R, ordering $\prec$ is *R-admissible* if each literal of the form $\forall \mathsf{inv}(R).A$ is $\prec$-minimal.

Throughout the rest of this paper we fix a "global" countably infinite set of *contexts* $\mathbf{X}$. Recall that in Section 8.1 we introduced Σ_L as the set of all literals (concepts of the form A, $\exists R.A$, $\forall R.A$); by $\Sigma_L^\exists$ we denote the set of all literals of the form $\exists R.A$. Finally, recall that we often treat conjunctions and disjunctions of literals as sets of their respective conjuncts and disjuncts. Sets $\mathbf{X}$, Σ_L, and $\Sigma_L^\exists$ provide us with building blocks for the following definition of a context structure; the notions of admissibility and covering were motivated at the end of Section 8.2.

Definition 8.3. A *context structure* is a tuple $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \prec \rangle$, where $\mathcal{V} \subseteq \mathbf{X}$ is a finite set of contexts, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \Sigma_L^\exists$ is a finite set of edges labeled by existential restrictions, function $\mathrm{core}\colon \mathcal{V} \to 2^{\Sigma_L}$ labels each context with a finite set of literals, and function $\prec$ assigns to each context $v \in \mathcal{V}$ a literal ordering $\prec_v$. Such $\mathcal{D}$ is *admissible* if, for each edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$, the literal ordering $\prec_u$ is R-admissible. Furthermore, $\mathcal{D}$ is *over* a set of literals $\mathbf{L}$ if all literals in $\mathcal{D}$ are contained in $\mathbf{L}$. Finally, given a context v, set $\mathrm{core}(v)$ is often treated as the conjunction of its atomic concepts, thus

allowing core(v) to occur in concepts and axioms.

Let $v \in \mathcal{V}$ be an arbitrary context of $\mathcal{D}$; then, v is *trivial* if core(v) = $\emptyset$ and $\prec_v = \emptyset$. In addition, let $K \sqsubseteq M$ be an arbitrary query; then, v is *sound* for $K \sqsubseteq M$ if core(v) $\subseteq K$; v is *complete* for $K \sqsubseteq M$ if M is $\prec_v$-minimal; and v *covers* $K \sqsubseteq M$ if v is both sound and complete for $K \sqsubseteq M$.

We next formalize the notion of a clause system as a family of sets of clauses indexed by the contexts. As we discussed in Section 8.2, the output of our consequence-based algorithm consists of a context structure and a clause system, and the latter can be used to decide the entailment of all relevant queries.

Definition 8.4. A *clause system* for a context structure $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \prec \rangle$ is a function $\mathcal{S}$ that assigns to each context $v \in \mathcal{V}$ a finite set of clauses $\mathcal{S}(v)$.

To simplify the presentation, in the rest of this section we fix a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$ and a finite set of queries $\mathcal{Q}$—that is, we assume that all subsequent definitions and theorems in this section are implicitly parameterized with $\mathcal{O}$ and $\mathcal{Q}$. Furthermore, we fix $\mathbf{L}$ to be the set of literals occurring in $\mathcal{O} \cup \mathcal{Q}$; since $\mathcal{O}$ and $\mathcal{Q}$ are finite sets, set $\mathbf{L}$ is finite as well.

As we explained in Section 8.2, our algorithm is based on the inference rules shown in Table 8.1. The Hyper, Succ, and Pred rules are responsible for completeness, and we discussed the intuitions behind these rules in Section 8.2. The Elim rule supports redundancy elimination and is not needed for completeness; we discuss the intuitions behind this rule in Section 8.4. The completeness of our algorithm is guaranteed by Theorem 8.5, the proof of which is given in Appendix D.1. The theorem essentially says that, given a context structure $\mathcal{D}$ and a clause system $\mathcal{S}$ saturated under the the Hyper, Pred, and Succ rules, we can read off the consequences of the form $K \sqsubseteq M$ from the sets $\mathcal{S}(v)$ provided that $K \sqsubseteq M$ is a query, the context v is complete (cf. Definition 8.3) for $K \sqsubseteq M$, and set $\mathcal{S}(v)$ is appropriately initialized. The initialization is similar to asserting $L(a)$ for each literal $L \in K$ at the beginning of a hypertableau test of $\mathcal{O} \models K \sqsubseteq M$. Note that Theorem 8.5 depends on the Succ rule being *not applicable*, and so the exact definition of strategy used in the postcondition of the Succ rule is not relevant for completeness. In the rest of this section we discuss how to initialize the sets $\mathcal{S}(v)$ and how to satisfy the precondition of the Succ rule to obtain a sound and complete algorithm.

Hyper	If	$\bigsqcap_{i=1}^{n} A_i \sqsubseteq M \in \mathcal{O}$,
		$K_i \sqsubseteq M_i \sqcup A_i \in \mathcal{S}(v)$ with $A_i \not\prec_v M_i$ for $1 \leq i \leq n$,
		and $\bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i \notin \mathcal{S}(v)$,
	then	add $\bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i$ to $\mathcal{S}(v)$.
Pred	If	$\langle v, u, \exists R.A \rangle \in \mathcal{E}$,
		$A \sqcap \bigsqcap_{i=1}^{n} B_i \sqsubseteq \bigsqcup_{j=1}^{m} \forall \mathsf{inv}(R).C_j \in \mathcal{S}(u)$ or $\bigsqcap_{i=1}^{n} B_i \sqsubseteq \bigsqcup_{j=1}^{m} \forall \mathsf{inv}(R).C_j \in \mathcal{S}(u)$,
		$K_0 \sqsubseteq M_0 \sqcup \exists R.A \in \mathcal{S}(v)$ with $\exists R.A \not\prec_v M_0$,
		$K_i \sqsubseteq M_i \sqcup \forall R.B_i \in \mathcal{S}(v)$ with $\forall R.B_i \not\prec_v M_i$ for $1 \leq i \leq n$,
		and $\bigsqcap_{i=0}^{n} K_i \sqsubseteq \bigsqcup_{i=0}^{n} M_i \sqcup \bigsqcup_{j=1}^{m} C_j \notin \mathcal{S}(v)$,
	then	add $\bigsqcap_{i=0}^{n} K_i \sqsubseteq \bigsqcup_{i=0}^{n} M_i \sqcup \bigsqcup_{j=1}^{m} C_j$ to $\mathcal{S}(v)$.
Succ	If	$K \sqsubseteq M \sqcup \exists R.A \in \mathcal{S}(v)$ with $\exists R.A \not\prec_v M$, and
		for $\mathbf{B}_p \coloneqq \{B \mid K' \sqsubseteq M' \sqcup \forall R.B \in \mathcal{S}(v) \text{ and } \forall R.B \not\prec_v M'\}$,
		no edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ exists such that $L \sqsubseteq L \hat{\in} \mathcal{S}(u)$ for each $L \in \{A\} \cup \mathbf{B}_p$,
	then	let $\langle u, \mathrm{core}', \prec' \rangle \coloneqq \mathsf{strategy}(\exists R.A, \mathbf{B}_k, \mathcal{D})$ where $\mathbf{B}_k \coloneqq \{B \mid \top \sqsubseteq \forall R.B \in \mathcal{S}(v)\}$;
		if $u \in \mathcal{V}$, then let $\prec_u \coloneqq \prec_u \cap \prec'$, and
		otherwise let $\mathcal{V} \coloneqq \mathcal{V} \cup \{u\}$, $\mathrm{core}(u) \coloneqq \mathrm{core}'$, $\prec_u \coloneqq \prec'$, and $\mathcal{S}(u) \coloneqq \{\top \sqsubseteq L \mid L \in \mathrm{core}(u)\}$;
		add $\langle v, u, \exists R.A \rangle$ to $\mathcal{E}$; and
		for each $L \in [\{A\} \cup \mathbf{B}_p] \setminus \mathrm{core}(v)$ such that $L \sqsubseteq L \notin \mathcal{S}(u)$, add $L \sqsubseteq L$ to $\mathcal{S}(u)$.
Elim	If	$K_1 \sqsubseteq M_1 \in \mathcal{S}(v)$,
		$K_2 \sqsubseteq M_2 \in \mathcal{S}(v)$, and
		$K_1 \sqsubseteq M_1$ is a strengthening of $K_2 \sqsubseteq M_2$ and the two clauses are distinct,
	then	remove $K_2 \sqsubseteq M_2$ from $\mathcal{S}(v)$.

Table 8.1: Consequence-based inference rules for $\mathcal{ALCI}$

Theorem 8.5 (Completeness)**.** *Let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \prec \rangle$ be an admissible context structure, and let $\mathcal{S}$ be a clause system for $\mathcal{D}$ such that the* Hyper*,* Pred*, and* Succ *rules from Table 8.1 are not applicable to $\mathcal{D}$ and $\mathcal{S}$. Then, $K \sqsubseteq M \hat{\in} \mathcal{S}(v)$ holds for each query $K \sqsubseteq M$ and each context $v \in \mathcal{V}$ that satisfy all of the following three conditions:*

- $\mathcal{O} \models K \sqsubseteq M$*,*
- *context v is complete for query $K \sqsubseteq M$, and*
- $K \sqsubseteq L \hat{\in} \mathcal{S}(v)$ *for each literal $L \in K$.*

As we discussed in Section 8.2, our framework is parameterized by a function strategy, which we formalize in Definition 8.6. Intuitively, when the Succ rule identifies a context v and an existential restriction $\exists R.A$ for which the rule's precondition is not satisfied, $\mathsf{strategy}(\exists R.A, \mathbf{B}_k, \mathcal{D})$ returns a triple $\langle u, \mathrm{core}', \prec' \rangle$ that specifies how to modify $\mathcal{D}$ and $\mathcal{S}$ so that the precondition becomes satisfied. The strategy has two options. First, the strategy can decide to extend $\mathcal{D}$ by returning a fresh context

u; in such a case, u is added to $\mathcal{D}$, and core$'$ and $\prec'$ specify how to initialize core(u) and $\prec_u$. Second, the strategy can decide to reuse a context u that is already a part of $\mathcal{D}$; if so, then ordering $\prec_u$ is intersected with $\prec'$ to ensure R-admissibility, but also preserve admissibility for all other roles. Definition 8.6 requires each strategy to be bounded: if the signature is finite, then the strategy should return only finitely many different results on all possible inputs. Please note that a finite signature does not necessarily bound the size of $\mathcal{D}$, so the number of different arguments (and return values) for strategy is not bounded. Bounded strategies can thus introduce only finitely many fresh contexts, which ensures termination.

Definition 8.6. An *expansion strategy* is a function strategy computable in polynomial time that takes a literal $\exists R.A$, a set of atomic concepts $\mathbf{B}_k$, and a context structure $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \prec \rangle$. The result of strategy($\exists R.A, \mathbf{B}_k, \mathcal{D}$) is a triple $\langle u, \text{core}', \prec' \rangle$ where

- core$'$ is a subset of $\{A\} \cup \mathbf{B}_k$,
- $\prec'$ is an R-admissible literal ordering, and
- either $u \in \mathbf{X} \setminus \mathcal{V}$ is a fresh context, or $u \in \mathcal{V}$ is a context in $\mathcal{D}$ such that core(u) = core$'$.

Each expansion strategy must be *bounded*: for each finite set of literals $\mathbf{L}$, the number $|\text{strategy}|_{\mathbf{L}}$ of different values that strategy can return for all possible arguments over $\mathbf{L}$ must be finite.

We next define when a context structure $\mathcal{D}$ and a clause system $\mathcal{S}$ are sound for $\mathcal{O}$. Recall that, for a context v, clause $K \sqsubseteq M$ in $\mathcal{S}(v)$ is "relative" to core(v)—that is, the clause should be interpreted as core(v) $\sqcap K \sqsubseteq M$. Soundness for a clause simply means that $\mathcal{O} \models \text{core}(v) \sqcap K \sqsubseteq M$ should hold for each clause $K \sqsubseteq M \in \mathcal{S}(v)$. To understand the notion of soundness for a context structure, consider an arbitrary edge $\langle v, u, \exists R.A \rangle$ in $\mathcal{D}$. As we explained in Section 8.2, the Pred rule "pulls" information from $\mathcal{S}(u)$ into $\mathcal{S}(v)$; however, the clauses in $\mathcal{S}(v)$ and $\mathcal{S}(u)$ are "relative" to core(v) and core(u), respectively, so core(u) and core(v) must be related as specified in Definition 8.7 to make the Pred rule sound. Please remember that core(u) and core(v) should be understood as conjunctions of the respective sets of atomic concepts.

Definition 8.7. A context structure $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \prec \rangle$ is *sound* for $\mathcal{O}$ if each edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ satisfies $\mathcal{O} \models \text{core}(v) \sqcap \exists R.A \sqsubseteq \exists R.[\text{core}(u) \sqcap A]$. Furthermore, a clause system $\mathcal{S}$ is *sound* for $\mathcal{O}$ if each context $v \in \mathcal{V}$ and each clause $K \sqsubseteq M \in \mathcal{S}(v)$ satisfy $\mathcal{O} \models \text{core}(v) \sqcap K \sqsubseteq M$.

Proposition 8.8 shows that the inference rules in Table 8.1 preserve admissibility and soundness of a context structure and a clause system. The former follows immediately from Definition 8.6, and the proof of the latter is analogous to the soundness proof for first-order resolution [18]: we show that each clause introduced by an inference rule is a logical consequence of $\mathcal{O}$.

Proposition 8.8 (Soundness)**.** *Let $\mathcal{D}_1 = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \prec \rangle$ be a context structure, let $\mathcal{S}_1$ be a clause system for $\mathcal{D}_1$, and let $\mathcal{D}_2$ and $\mathcal{S}_2$ be the context structure and the clause system, respectively, obtained by applying an inference rule from Table 8.1 to $\mathcal{D}_1$ and $\mathcal{S}_1$. If $\mathcal{D}_1$ is admissible, then $\mathcal{D}_2$ is admissible. Furthermore, if both $\mathcal{D}_1$ and $\mathcal{S}_1$ are sound for $\mathcal{O}$, then both $\mathcal{D}_2$ and $\mathcal{S}_2$ are sound for $\mathcal{O}$.*

Proof. Assume that $\mathcal{D}_1$ is admissible and that the Succ rule is applied to $\mathcal{D}_1$ as shown in Table 8.1. By Definition 8.6, literal ordering $\prec'$ is R-admissible; furthermore, for arbitrary literal orderings $\prec_1$ and $\prec_2$ that are R- and S-admissible, respectively, set $\prec_1 \cap \prec_2$ is a literal ordering that is both R- and S-admissible. Therefore, regardless of whether context u is fresh or not, $\mathcal{D}_2$ is admissible. Now assume that both $\mathcal{D}_1$ and $\mathcal{S}_1$ are sound; we next show that both $\mathcal{D}_2$ and $\mathcal{S}_2$ are sound as well. To this end, let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ be an arbitrary model of $\mathcal{O}$, and consider all possible inference rules that derive a clause in $\mathcal{S}_2$ or modify $\mathcal{D}_2$. In each case we assume that the rule is applied as in Table 8.1

(Hyper rule) We will show that $\mathcal{O} \models \mathrm{core}(v) \sqcap \bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i$. To this end, consider an arbitrary element $\delta \in \Delta^{\mathcal{I}}$ such that $\delta \in (\mathrm{core}(v) \sqcap \bigsqcap_{i=1}^{n} K_i)^{\mathcal{I}}$. Now $\mathcal{S}_1$ is sound for $\mathcal{O}$ so, for each $1 \leq i \leq n$, we have $\mathcal{O} \models \mathrm{core}(v) \sqcap K_i \sqsubseteq M_i \sqcup A_i$, which implies $\delta \in (M_i \sqcup A_i)^{\mathcal{I}}$. If $\delta \in M_i^{\mathcal{I}}$ for some $1 \leq i \leq n$, then clearly $\delta \in (\bigsqcup_{i=1}^{n} M_i)^{\mathcal{I}}$. Otherwise, we have $\delta \in (\bigsqcap_{i=1}^{n} A_i)^{\mathcal{I}}$, but then $\bigsqcap_{i=1}^{n} A_i \sqsubseteq M \in \mathcal{O}$ implies $\delta \in M^{\mathcal{I}}$. In either case, we have $\delta \in (M \sqcup \bigsqcup_{i=1}^{n} M_i)^{\mathcal{I}}$. Since δ was chosen arbitrarily, $\mathcal{I}$ satisfies the conclusion of the rule, as required.

(Pred rule) We will show that $\mathcal{O} \models \mathrm{core}(v) \sqcap \bigsqcap_{i=0}^{n} K_i \sqsubseteq \bigsqcup_{i=0}^{n} M_i \sqcup \bigsqcup_{j=1}^{m} C_j$. To this end, consider an arbitrary element $\delta \in \Delta^{\mathcal{I}}$ such that $\delta \in (\mathrm{core}(v) \sqcap \bigsqcap_{i=0}^{n} K_i)^{\mathcal{I}}$. If $\delta \in M_i^{\mathcal{I}}$ for some $0 \leq i \leq n$, then clearly $\delta \in (\bigsqcup_{i=0}^{n} M_i)^{\mathcal{I}}$. Otherwise, as $\mathcal{S}_1$ is sound for $\mathcal{O}$, we have $\delta \in (\exists R.A)^{\mathcal{I}}$ and $\delta \in (\forall R.B_i)^{\mathcal{I}}$ for each $1 \leq i \leq n$. Since $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ and $\mathcal{D}_1$ is sound, $\mathcal{O} \models \mathrm{core}(v) \sqcap \exists R.A \sqsubseteq \exists R.[\mathrm{core}(u) \sqcap A]$; therefore, since $\delta \in [\mathrm{core}(v) \sqcap \exists R.A]^{\mathcal{I}}$, we have $\delta \in (\exists R.[\mathrm{core}(u) \sqcap A])^{\mathcal{I}}$. Hence, an element $\gamma \in \Delta^{\mathcal{I}}$ exists such that $\langle \delta, \gamma \rangle \in R^{\mathcal{I}}$ and $\gamma \in [\mathrm{core}(u) \sqcap A \sqcap \bigsqcap_{i=1}^{n} B_i]^{\mathcal{I}}$. But then, due to either of the two alternative premises of the rule, we also have $\gamma \in [\bigsqcup_{j=1}^{m} \forall \mathsf{inv}(R).C_j]^{\mathcal{I}}$; hence $\delta \in (\bigsqcup_{j=1}^{m} C_j)^{\mathcal{I}}$. Since δ was chosen arbitrarily, $\mathcal{I}$ satisfies the conclusion of the rule, as required.

(Succ rule) The rule introduces only clauses of the form $L \sqsubseteq L$, and $\mathcal{I} \models \mathrm{core}(u) \sqcap L \sqsubseteq L$ clearly holds. We next show that the new edge $\langle v, u, \exists R.A \rangle$ introduced by the rule satisfies the condition in Definition 8.7. For each atomic concept $B \in \mathbf{B}_k$, we have $\top \sqsubseteq \forall R.B \in \mathcal{S}(v)$ by the rule preconditions, so $\mathcal{O} \models \mathrm{core}(v) \sqsubseteq \forall R.B$ as $\mathcal{S}_1$ is sound for $\mathcal{O}$; but then, we have $\mathcal{O} \models \mathrm{core}(v) \sqcap \exists R.A \sqsubseteq \exists R.[A \sqcap \mathbf{B}_k]$. Finally, since $\mathrm{core}(u) = \mathrm{core}' \subseteq A \sqcap \mathbf{B}_k$, we also have $\mathcal{O} \models \mathrm{core}(v) \sqcap \exists R.A \sqsubseteq \exists R.[\mathrm{core}(u) \sqcap A]$, as required. □

The following notion of a covering mapping maps each query $q \in \mathcal{Q}$ to a context in a context structure. Intuitively, our algorithm will use such a mapping to find for each query $q \in \mathcal{Q}$ a context that can be used to verify $\mathcal{O} \models q$.

Definition 8.9. Let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \prec \rangle$ be a context structure. A *covering mapping* from $\mathcal{Q}$ to $\mathcal{D}$ is a function $\vartheta : \mathcal{Q} \to \mathcal{V}$ such that each query $q \in \mathcal{Q}$ is covered in the context $\vartheta(q)$.

We are finally ready to formally define our consequence-based reasoning algorithm. Apart from $\mathcal{O}$ and $\mathcal{Q}$, the algorithm is parameterized by an expansion strategy `strategy`, a context structure $\mathcal{D}$ with no edges, and a covering mapping ϑ. By passing $\mathcal{D}$ and ϑ as arguments, the algorithm's users can decide how to initialize the contexts necessary for checking the queries in $\mathcal{Q}$; we discuss reasonable possibilities in Section 8.5.1. Since the algorithm is given a context structure without edges, $\mathcal{D}$ is trivially sound for $\mathcal{O}$; by inductively applying Proposition 8.8, the resulting clause system is sound for $\mathcal{O}$. Furthermore, the algorithm is complete because steps 1 and 2 satisfy the preconditions of Theorem 8.5.

Algorithm 8.10. The *consequence-based algorithm* for $\mathcal{ALCI}$ takes as input $\mathcal{O}$, $\mathcal{Q}$, a context structure $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \prec \rangle$ over $\mathbf{L}$ with $\mathcal{E} = \emptyset$, an expansion strategy `strategy`, and a covering mapping ϑ from $\mathcal{Q}$ to $\mathcal{D}$. The algorithm (nondeterministically) extends $\mathcal{D}$ and computes a clause system $\mathcal{S}$ for the extended context structure as follows.

1. Set $\mathcal{S}(v) := \{\top \sqsubseteq L \mid L \in \mathrm{core}(v)\}$ for each context $v \in \mathcal{V}$.
2. For each query $K \sqsubseteq M \in \mathcal{Q}$, let $v := \vartheta(K \sqsubseteq M)$; then, for each literal $L \in K \setminus \mathrm{core}(v)$, add $L \sqsubseteq L$ to $\mathcal{S}(v)$.
3. Exhaustively apply inference rules from Table 8.1.

Please note that Algorithm 8.10 is nondeterministic: the conclusion of an inference rule is added to $\mathcal{S}(v)$ only if $\mathcal{S}(v)$ does not contain a strengthening of the conclusion, and the order of inference rule application is not predetermined; therefore, the resulting $\mathcal{D}$ and $\mathcal{S}$ are not unique. This, however, is don't-care nondeterminism: any order of inference rule applications suffices for soundness and completeness.

In Proposition 8.12 we formalize the soundness and completeness argument outlined earlier. Towards this goal, in Lemma 8.11 we show that the Elim rule never deletes the "relevant" consequences derived by our algorithm: whenever a clause $K \sqsubseteq M$ is deleted from some set $\mathcal{S}(v)$, after deletion the set still contains a strengthening of the deleted clause.

Lemma 8.11. *If, at some point in the execution of Algorithm 8.10, $K \sqsubseteq M \mathrel{\hat{\in}} \mathcal{S}(v)$ holds for some clause $K \sqsubseteq M$ and context v, then $K \sqsubseteq M \mathrel{\hat{\in}} \mathcal{S}(v)$ holds at all future points as well.*

Proof. The proof is by a straightforward induction on the application of the rules. In particular, rules Hyper, Pred, and Succ just add clauses, so their application clearly preserves this property. Furthermore, the Elim rule removes a clause $K_2 \sqsubseteq M_2$ from $\mathcal{S}(v)$ only if $K_1 \sqsubseteq M_1 \in \mathcal{S}(v)$ where $K_1 \sqsubseteq M_1$ is a strengthening of $K_2 \sqsubseteq M_2$; since the strengthening relation on clauses is transitive, an application of the Elim rule clearly preserves this property as well. □

Proposition 8.12. *Let $\mathcal{S}$ be a clause system obtained by applying Algorithm 8.10 to $\mathcal{O}$, $\mathcal{Q}$, a context structure $\mathcal{D}$, an expansion strategy* strategy*, and a covering mapping ϑ. Then, for each query $q \in \mathcal{Q}$, we have $\mathcal{O} \models q$ if and only if $q \mathrel{\hat{\in}} \mathcal{S}(\vartheta(q))$.*

Proof. Consider an arbitrary query $K \sqsubseteq M \in \mathcal{Q}$ and let $v = \vartheta(K \sqsubseteq M)$. Context v covers $K \sqsubseteq M$, so v is sound for $K \sqsubseteq M$ and thus $\mathrm{core}(v) \subseteq K$ holds by Definition 8.3, and v is also complete for $K \sqsubseteq M$.

Assume that $K \sqsubseteq M \mathrel{\hat{\in}} \mathcal{S}(v)$, so a clause $K' \sqsubseteq M' \in \mathcal{S}(v)$ exists such that $K' \subseteq K$ and $M' \subseteq M$. For each clause $\top \sqsubseteq L$ added to some $\mathcal{S}(u)$ in step 1, we have $\mathcal{O} \models \mathrm{core}(u) \sqsubseteq L$ since $L \in \mathrm{core}(u)$; furthermore, for each clause $L \sqsubseteq L$ added to $\mathcal{S}(u)$ in step 2, we clearly have $\mathcal{O} \models \mathrm{core}(u) \sqcap L \sqsubseteq L$; finally, Algorithm 8.10 is applied to a sound context structure, so $\mathcal{D}$ and $\mathcal{S}$ are sound for $\mathcal{O}$ by Proposition 8.8, and $\mathcal{O} \models \mathrm{core}(v) \sqcap K' \sqsubseteq M'$ holds. Finally, $K' \cup \mathrm{core}(v) \subseteq K$ and $M' \subseteq M$ imply $\mathcal{O} \models K \sqsubseteq M$, as required.

Conversely, assume that $\mathcal{O} \models K \sqsubseteq M$. Consider an arbitrary literal $L \in K$; if $L \in \mathrm{core}(v)$, then $\top \sqsubseteq L$ is added to $\mathcal{S}(v)$ in step 1, and if $L \in K \setminus \mathrm{core}(v)$, then $L \sqsubseteq L$ is added to $\mathcal{S}(v)$ in step 2; in either case, we have that $K \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ holds after step 2, and Lemma 8.11 ensures that this property is preserved during the algorithm's execution. But then, since v is complete for $K \sqsubseteq M$, we have $K \sqsubseteq M \mathrel{\hat{\in}} \mathcal{S}(v)$ by Theorem 8.5, as required. □

Finally, we determine the complexity of our algorithm.

Proposition 8.13 (Termination)**.** *When applied to $\mathcal{O}$, $\mathcal{Q}$, a context structure $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \prec \rangle$, an expansion strategy* strategy*, and a covering mapping ϑ, Algorithm 8.10 terminates in time polynomial in $4^{|\mathbf{L}|^2}$, $|\mathsf{strategy}|_{\mathbf{L}} + |\mathcal{V}|$, and $\|\mathcal{O}\| + \|\mathcal{Q}\|$.*

Proof. When applied to arguments over $\mathbf{L}$, the result of the expansion strategy is clearly over $\mathbf{L}$ as well; therefore, the context structure and the clause system computed using Algorithm 8.10 are over $\mathbf{L}$. The number of different clauses over $\mathbf{L}$ is bounded by $c = 2^{|\mathbf{L}|} \cdot 2^{|\mathbf{L}|} = 4^{|\mathbf{L}|}$ since each literal from $\mathbf{L}$ can independently occur in the clause antecedent and/or the clause consequent. Algorithm 8.10 is applied to a context structure with $|\mathcal{V}|$ contexts, and the expansion strategy can introduce at most $|\mathsf{strategy}|_{\mathbf{L}}$ additional contexts; hence, the total number of contexts introduced in the algorithm is bounded by $m = |\mathsf{strategy}|_{\mathbf{L}} + |\mathcal{V}|$.

We call all objects needed to apply an inference rule from Table 8.1 *premises*; for the Succ rule, this includes set $\mathbf{B}_p$. We next show that, for the Hyper, Pred, and Succ rules, the number of different premises is polynomial in $c^{|\mathbf{L}|}$, m, and $\|\mathcal{O}\| + \|\mathcal{Q}\|$; recall that $|\mathbf{L}| \leq \|\mathcal{O}\| + \|\mathcal{Q}\|$. For the Hyper rule, context $v \in \mathcal{V}$ can be chosen in at most m ways, clause $\bigsqcap_i A_i \sqsubseteq M \in \mathcal{O}$ can be chosen in at most $|\mathcal{O}|$ ways, and each of the $n \leq |\mathbf{L}|$ clauses $K_i \sqsubseteq M_i \sqcup A_i \in \mathcal{S}(v)$ can be chosen in at most c ways, so there are at most $c^{|\mathbf{L}|}$ ways of choosing such a set of clauses. For the Pred rule, edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ can be chosen in at most $m^2 \cdot |\mathbf{L}|$ ways, clause $A \sqcap \bigsqcap_i B_i \sqsubseteq \bigsqcup_j \forall \mathsf{inv}(R).C_j \in \mathcal{S}(u)$ or $\bigsqcap_i B_i \sqsubseteq \bigsqcup_j \forall \mathsf{inv}(R).C_j \in \mathcal{S}(u)$ can be chosen in at most c ways, and each of the $n + 1 \leq |\mathbf{L}|$ clauses $K_0 \sqsubseteq M_0 \sqcup \exists R.A \in \mathcal{S}(v)$ and $K_i \sqsubseteq M_i \sqcup \forall R.B_i \in \mathcal{S}(v)$ can be chosen in at most c ways, so there are at most $c^{|\mathbf{L}|}$ ways of choosing such a set of clauses. For the Succ rule, context $v \in \mathcal{V}$ can be chosen in at most m ways, literal $\exists R.A$ can be chosen in at most $|\mathbf{L}|$ ways, clause $K \sqsubseteq M \sqcup \exists R.A \in \mathcal{S}(v)$ can be chosen in at most c ways, and there are at most $2^{|\mathbf{L}|}$ different sets $\mathbf{B}_p$.

Each rule in Table 8.1 adds a clause $K \sqsubseteq M$ to some $\mathcal{S}(v)$ only if $K \sqsubseteq M \hat{\notin} \mathcal{S}(v)$; thus, due to Lemma 8.11, each such $K \sqsubseteq M$ can be added to some $\mathcal{S}(v)$ at most once; but then, the Elim rule can eliminate such $K \sqsubseteq M$ from $\mathcal{S}(v)$ at most once as well. Thus, no inference rule is applied twice to the same premises, so the number of inference rule applications is bounded by a number that is polynomial in $c^{|\mathbf{L}|}$, m, and $\|\mathcal{O}\| + \|\mathcal{Q}\|$. Moreover, strategy is computable in polynomial time, so each inference rule can be applied to fixed premises in polynomial time. Consequently, the algorithm can be implemented so that it runs in time polynomial in $c^{|\mathbf{L}|}$, m, and $\|\mathcal{O}\| + \|\mathcal{Q}\|$. □

8.4 Redundancy Elimination

Resolution can often derive *redundant* clauses—that is, clauses that are not necessary for a proof. Such clauses can give rise to a large number of other redundant clauses, so it is beneficial to detect and eliminate redundant clauses whenever possible. To this end, modern first-order theorem provers employ a number of *redundancy elimination rules* that eliminate or simplify certain clauses; Weidenbach [138] presents an overview of these techniques. Inspired by redundancy elimination in first-order theorem provers, we equipped our consequence-based framework with analogous possibilities, which we discuss in this section.

First, the completeness Theorem 8.5 only requires each set of clauses $\mathcal{S}(v)$ to contain a strengthening of each conclusion obtained by applying the inference rules from Table 8.1; therefore, our inference rules derive a conclusion in $\mathcal{S}(v)$ only if $\mathcal{S}(v)$ does not already contain a strengthening of the conclusion. This is analogous to *forward subsumption* in first-order theorem proving: a newly derived clause is kept only if the clause is not redundant given the clauses derived thus far.

Second, the Elim rule deletes a clause $K_2 \sqsubseteq M_2$ from $\mathcal{S}(v)$ if $\mathcal{S}(v)$ contains a clause $K_1 \sqsubseteq M_1$ such that $K_1 \sqsubseteq M_1$ is a strengthening of $K_2 \sqsubseteq M_2$ and the two clauses are distinct. Note that $K_2 \sqsubseteq M_2$ can be derived before $K_1 \sqsubseteq M_1$, in which case the technique described in the previous paragraph will not eliminate $K_2 \sqsubseteq M_2$, and so $K_2 \sqsubseteq M_2$ can potentially participate in further redundant inferences. The Elim rule is thus analogous to *backward subsumption* in first-order theorem proving.

In the rest of this section we discuss certain aspects of our redundancy elimination techniques. We first consider *syntactic tautologies*, which are clauses of the form $K \sqcap L \sqsubseteq M \sqcup L$ with either K or M (or both) nonempty; note that this definition excludes clauses of the form $L \sqsubseteq L$, which

are needed for completeness in Theorem 8.5 and are thus not redundant. Since our algorithm never derives a clause whose strengthening is already present in the context, the following Proposition 8.14 implies that the algorithm never derives syntactic tautologies.

Proposition 8.14. *If, at some point in the execution of Algorithm 8.10, $K \sqsubseteq M \mathrel{\hat{\in}} \mathcal{S}(v)$ holds for some clause $K \sqsubseteq M$ and context v, then $L \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ also holds at this point for each literal $L \in K$.*

Proof. The proof is by induction on the application of inference rules. Each clause introduced in steps 1 and 2 of Algorithm 8.10 or by the Succ rule obviously satisfies this property. Furthermore, an application of the Elim rule preserves this property by Lemma 8.11. Now consider an application of the Hyper or the Pred rule as shown in Table 8.1; then, for each literal L occurring in the antecedent of the conclusion, integer i exists such that $L \in K_i$; but then, the premise corresponding to K_i satisfies this property by the induction assumption, so $L \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ holds. □

The Elim rule can also be used to define more powerful redundancy elimination rules. Assume that, for some context v, set $\mathcal{S}(v)$ contains clauses $K_1 \sqsubseteq M_1 \sqcup L$ and $K_2 \sqcap L \sqsubseteq M_2$ with $K_1 \subseteq K_2$ and $M_1 \subseteq M_2$. Clause $K_1 \sqcap K_2 \sqsubseteq M_1 \sqcup M_2 = K_2 \sqsubseteq M_2$ is a logical consequence of $\mathcal{S}(v)$; but then, although this clause is not derived by an inference rule from Table 8.1 (in particular, note that the Hyper rule *does not* derive this clause), extending $\mathcal{S}(v)$ with $K_2 \sqsubseteq M_2$ is sound. Thus, if we add $K_2 \sqsubseteq M_2$ to $\mathcal{S}(v)$, we can delete clause $K_2 \sqcap L \sqsubseteq M_2$ from $\mathcal{S}(v)$ using the Elim rule; essentially, we thus obtain a simplification rule that eliminates L from clause $K_2 \sqcap L \sqsubseteq M_2$. Analogously, if the clauses satisfy $K_2 \subseteq K_1$ and $M_2 \subseteq M_1$, then we can eliminate L from clause $K_1 \sqsubseteq M_1 \sqcup L$. This is closely related to the *contextual literal cutting* rule used in the first-order prover E [114].

Redundancy elimination may be difficult to implement fully in practice: even with complex index structures, checking whether some $\mathcal{S}(v)$ contains a strengthening of some clause may be inefficient. Please note, however, that both forms of redundancy elimination are *optional*—that is, they are not needed for completeness. Therefore, instead of $\hat{\in}$, one can use $\in$ to check rule applicability, and one does not need to apply the Elim rule exhaustively. Furthermore, redundancy checking can be restricted to clauses of the form $\top \sqsubseteq A$ and $L \sqsubseteq L$; since these clauses contain only one literal, they can be indexed using a simple hash table. Finally, an advanced implementation can "switch on" redundancy elimination only if set $\mathcal{S}(v)$ grows beyond a certain size. In most cases, however, eliminating syntactic tautologies should not cause any noticeable overhead.

8.5 Initialization and Expansion Strategies

We next discuss ways of instantiating our consequence-based framework presented in Section 8.3. In particular, in Section 8.5.1 we discuss possibilities for initializing the context structure and the covering mapping that are given to Algorithm 8.10. Then, in Section 8.5.2 we discuss possible expansion strategies.

8.5.1 Initializing the Context Structure and the Covering Mapping

Algorithm 8.10 takes as input a context structure $\mathcal{D}$ without any edges and a covering mapping ϑ, and so the initialization of $\mathcal{D}$ and ϑ is deferred to the algorithm's users. We next discuss several reasonable approaches to initialization.

First possibility is to introduce just one trivial (cf. Definition 8.3) context v_0. In such a case, step 2 of Algorithm 8.10 initializes $\mathcal{S}(v_0)$ with many clauses $L \sqsubseteq L$; this may give rise to a large number of inferences at context v_0, and may be further exacerbated by the fact that $\prec_{v_0}$ is empty. Hence, such an approach is unlikely to be efficient apart from in very simple cases.

Second possibility is to introduce a separate context $v_q = \vartheta(q)$ for each query $q = K \sqsubseteq M \in \mathcal{Q}$, define $\mathrm{core}(v_q) \coloneqq K$, and define $\prec_{v_q}$ such that M is $\prec_{v_q}$-minimal and the remaining literals are ordered arbitrarily. This has the opposite effect from the first approach: step 2 of Algorithm 8.10 initializes each set $\mathcal{S}(v_q)$ with the least possible number of clauses $L \sqsubseteq L$, and ordering $\prec_{v_q}$ is as fine-grained as possible, which maximally reduces the number of inferences at context v_q. The main drawback, however, is that the number of contexts needed is linear in the size of $\mathcal{Q}$.

Third possibility is to introduce a context v_K for each conjunction K occurring in the antecedent of some query in $\mathcal{Q}$, define $\mathrm{core}(v_K) \coloneqq K$, and define $\prec_{v_K}$ so that each literal occurring in the consequent of some query in $\mathcal{Q}$ is $\prec_{v_K}$-minimal while all remaining literals are ordered arbitrarily. We thus strike a balance between the number of contexts and the inferences performed.

Description logic reasoners are often used to classify an ontology $\mathcal{O}$, in which case $\mathcal{Q}$ contains $A \sqsubseteq B$ for all pairs of atomic concepts A and B. The second possibility then requires a quadratic number of contexts, which can be prohibitively large even on ontologies of moderate size. In contrast, the third possibility requires all atomic concepts in $\mathcal{O}$ to be minimal, which essentially "turns off" most ordering constraints. To address the former problem in hypertableau-based reasoners,

Glimm et al. [39] developed a classification algorithm that gathers information about known and possible subsumptions from the ABoxes encountered during the algorithm's execution in order to reduce the number of pairs of atomic concepts for which $\mathcal{O} \models A \sqsubseteq B$ needs to be checked. This algorithm can be adapted to the consequence-based framework as well. In particular, let $\mathcal{S}$ be a clause system produced by Algorithm 8.10 such that $\top \sqsubseteq \bot \mathrel{\hat{\notin}} \mathcal{S}(v)$ holds for each context v—that is, $\mathcal{O}$ is satisfiable. Then, if a context v exists such that $\mathrm{core}(v) \subseteq \{A\}$ and $A \sqsubseteq B \mathrel{\hat{\in}} \mathcal{S}(v)$, by the soundness of our algorithm we have that $\mathcal{O} \models A \sqsubseteq B$—in other words, subsumption $A \sqsubseteq B$ is *known*. Furthermore, Proposition 8.15, shown below, tells us how to identify subsumptions that are *not possible*. By tightly integrating the consequence-based algorithm with the appropriately modified classification algorithm, we can extend $\mathcal{Q}$ as needed, thus considerably reducing the size of this set. Furthermore, whenever the classification algorithm adds a query $A \sqsubseteq B$ to $\mathcal{Q}$, we have two options: we can introduce a fresh context $v_{A \sqsubseteq B}$; or we can introduce or reuse context v_A, keeping in mind that, whenever we reuse an existing context, we must refine the literal ordering $\prec_{v_A}$ so that B becomes $\prec_{v_A}$-minimal.

Proposition 8.15. *Let $\mathcal{S}$ be a clause system obtained by applying Algorithm 8.10 to a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$ and a finite set of queries $\mathcal{Q}$. Then, $\mathcal{O} \not\models A \sqsubseteq B$ holds for all atomic concepts A and B for which there exists a context v such that $A \sqsubseteq A \mathrel{\hat{\in}} \mathcal{S}(v)$, $A \sqsubseteq B \mathrel{\hat{\notin}} \mathcal{S}(v)$, and no clause of the form $K \sqsubseteq M \sqcup B \in \mathcal{S}(v)$ exists that satisfies $K \subseteq \{A\}$ and $B \not\prec_v M$.*

Proof. The proof of this proposition relies on the proof of Theorem 8.5 presented in Appendix D.1, the notion of pre-models introduced in Section D.1.1, and the symbol $\vdash$ introduced in Section D.1.4.

Let $\mathcal{S}$, A, B, and v be as specified in the proposition. Then, $A \sqsubseteq B \mathrel{\hat{\notin}} \mathcal{S}(v)$ clearly implies $A \sqsubseteq \bot \mathrel{\hat{\notin}} \mathcal{S}(v)$; moreover, $\bot$ is empty and therefore $\prec_v$-minimal; together with $A \sqsubseteq A \mathrel{\hat{\in}} \mathcal{S}(v)$, these observations imply $v \nvdash A \sqsubseteq \bot$. Thus, the construction of a pre-model $I = \langle \Delta, E, J \rangle$ in Section D.1.4 introduces an element $\delta^v_{A \sqsubseteq \bot} \in \Delta$ whose literal interpretation $J(\delta^v_{A \sqsubseteq \bot})$ is constructed as specified in Section D.1.2. Now consider an arbitrary clause $K \sqsubseteq M \sqcup B \in \mathcal{S}(v)$: by our assumption, at least one of $K \subseteq \{A\}$ or $B \not\prec_v M$ does not hold; thus, the conditions of Lemma D.6 are not satisfied, and so clause $K \sqsubseteq M \sqcup B$ is not productive. Since this holds for all clauses in $\mathcal{S}(v)$ with B in the consequent, we have $B \notin J(\delta^v_{A \sqsubseteq \bot})$. Furthermore, due to $A \sqsubseteq A \mathrel{\hat{\in}} \mathcal{S}(v)$ and Lemma D.7, we have $A \in J(\delta^v_{A \sqsubseteq \bot})$. Thus, we have $I \not\models A \sqsubseteq B$, and so $\mathcal{O} \not\models A \sqsubseteq B$ holds by Corollary D.3 and Claim D.14. □

8.5.2 Expansion Strategies

In this section we discuss reasonable expansion strategies. We use the ontology $\mathcal{O}$ and query q defined in the following example to demonstrate how the choice of strategy affects the inferences of our algorithm.

Example 8.16. Let $\mathcal{O}$ be the ontology containing axioms (8.31)–(8.38), and let $q = A \sqsubseteq E$. Clearly, $\mathcal{O} \models q$ holds due to (8.31)–(8.33).

$$A \sqsubseteq \exists R.B \quad (8.31) \qquad A \sqsubseteq \exists S.B \quad (8.34) \qquad A \sqcap B \sqsubseteq \exists T.F_1 \quad (8.36)$$

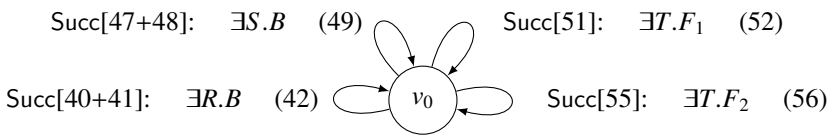
$$A \sqsubseteq \forall R.C_1 \quad (8.32) \qquad A \sqsubseteq D \sqcup \forall S.C_2 \quad (8.35) \qquad C_1 \sqcap C_2 \sqsubseteq C \quad (8.37)$$

$$B \sqcap C_1 \sqsubseteq \forall R^-.E \quad (8.33) \qquad\qquad\qquad B \sqcap C \sqsubseteq \exists T.F_2 \quad (8.38)$$

Trivial Strategy: The simplest possibility is to always reuse the trivial context v_0; thus, we define $\mathsf{trivial}(\exists R.A, \mathbf{B}_k, \mathcal{D}) = \langle v_0, \emptyset, \emptyset \rangle$; this strategy makes most sense if the trivial context v_0 is also used for initialization. The inferences of our algorithm on $\mathcal{O}$ with the trivial strategy are shown in Figure 8.2. As one can see, the algorithm is prolific: inferences (47)–(57) are not needed in order to derive (46). Such an algorithm resembles various resolution-based decision procedures for DLs [18].

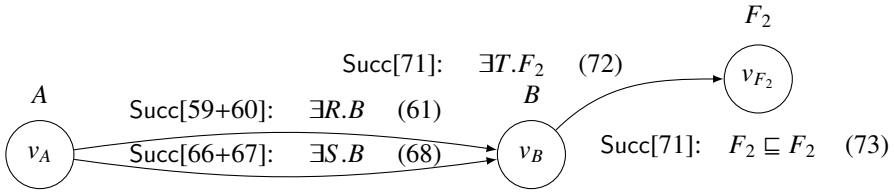
Cautious Strategy: To reduce the number of irrelevant inferences, we can introduce a context v_A for each atomic concept A, and define $\mathsf{cautious}(\exists R.A, \mathbf{B}_k, \mathcal{D}) = \langle v_A, \{A\}, \prec \rangle$, where $\prec$ is an arbitrary R-admissible ordering. The number of contexts is then limited to the number of atomic concepts occurring in $\mathcal{O}$ and $\mathcal{Q}$. Please remember that, whenever context v_A is reused, its literal ordering must be refined so that it stays admissible for each relevant role. As one can see in Figure 8.3, the cautious strategy considerably reduces the inferences of our algorithm; for example, it does not derive clauses containing $\exists T.F_1$. Nevertheless, due to the reuse of context v_B to satisfy existential restrictions (59) and (66), the algorithm derives clause (70), which then leads to the derivation of (71) and the introduction of context v_{F_2}.

Eager Strategy: To introduce even more contexts, we can consider a context $v_{\mathbf{A}}$ for each set of atomic concepts $\mathbf{A}$, and define $\mathsf{eager}(\exists R.A, \mathbf{B}_k, \mathcal{D}) = \langle v_{\{A\} \cup \mathbf{B}_k}, \{A\} \cup \mathbf{B}_k, \prec \rangle$, where $\prec$ is an arbitrary R-admissible ordering. As one can see in Figure 8.4, the algorithm now does not even derive clauses containing $\exists T.F_2$ and is considerably more efficient than in the previous two cases. In general,



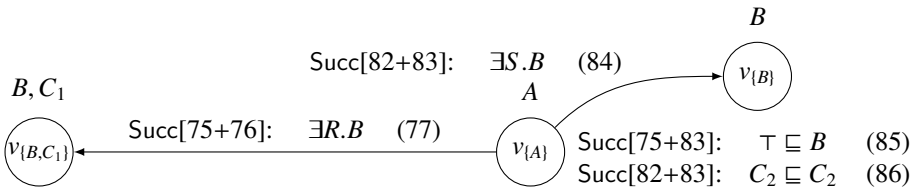
Initialization:	$A \sqsubseteq A$	(39)
Hyper[8.31+39]:	$A \sqsubseteq \exists R.B$	(40)
Hyper[8.32+39]:	$A \sqsubseteq \forall R.C_1$	(41)
Succ[40+41]:	$B \sqsubseteq B$	(43)
Succ[40+41]:	$C_1 \sqsubseteq C_1$	(44)
Hyper[8.33+43+44]:	$B \sqcap C_1 \sqsubseteq \forall R^-.E$	(45)
Pred[40+41+45]:	$A \sqsubseteq E$	(46)
Hyper[8.34+39]:	$A \sqsubseteq \exists S.B$	(47)
Hyper[8.35+39]:	$A \sqsubseteq D \sqcup \forall S.C_2$	(48)
Succ[47+48]:	$C_2 \sqsubseteq C_2$	(50)
Hyper[8.36+39+43]:	$A \sqcap B \sqsubseteq \exists T.F_1$	(51)
Succ[51]:	$F_1 \sqsubseteq F_1$	(53)
Hyper[8.37+44+50]:	$C_1 \sqcap C_2 \sqsubseteq C$	(54)
Hyper[8.38+43+44+50]:	$B \sqcap C_1 \sqcap C_2 \sqsubseteq \exists T.F_2$	(55)
Succ[55]:	$F_2 \sqsubseteq F_2$	(57)

Figure 8.2: Inferences with the trivial strategy



Initialization:	$\top \sqsubseteq A$	(58)	Succ[59+60]:	$\top \sqsubseteq B$	(62)
Hyper[8.31+58]:	$\top \sqsubseteq \exists R.B$	(59)	Succ[59+60]:	$C_1 \sqsubseteq C_1$	(63)
Hyper[8.32+58]:	$\top \sqsubseteq \forall R.C_1$	(60)	Hyper[8.33+62+63]:	$C_1 \sqsubseteq \forall R^-.E$	(64)
Pred[59+60+64]:	$\top \sqsubseteq E$	(65)	Succ[66+67]:	$C_2 \sqsubseteq C_2$	(69)
Hyper[8.34+58]:	$\top \sqsubseteq \exists S.B$	(66)	Hyper[8.37+63+69]:	$C_1 \sqcap C_2 \sqsubseteq C$	(70)
Hyper[8.35+58]:	$\top \sqsubseteq D \sqcup \forall S.C_2$	(67)	Hyper[8.38+62+70]:	$C_1 \sqcap C_2 \sqsubseteq \exists T.F_2$	(71)

Figure 8.3: Inferences with the cautious strategy



Succ[75+76]:	$\top \sqsubseteq B$	(78)	Initialization:	$\top \sqsubseteq A$	(74)
Succ[75+76]:	$\top \sqsubseteq C_1$	(79)	Hyper[8.31+74]:	$\top \sqsubseteq \exists R.B$	(75)
Hyper[8.33+78+79]:	$\top \sqsubseteq \forall R^-.E$	(80)	Hyper[8.32+74]:	$\top \sqsubseteq \forall R.C_1$	(76)
			Pred[75+80]:	$\top \sqsubseteq E$	(81)
			Hyper[8.34+74]:	$\top \sqsubseteq \exists S.B$	(82)
			Hyper[8.35+74]:	$\top \sqsubseteq D \sqcup \forall S.C_2$	(83)

Figure 8.4: Inferences with the eager strategy

however, the eager strategy can be problematic because the number of introduced contexts can be exponential in the number of atomic concepts occurring in $\mathcal{O}$ and $\mathcal{Q}$.

Refined Strategies: One can refine all of the strategies discussed above by using a different context per role R. For example, the cautious strategy can be refined to use a context v_A^R for each atomic concept A and role R; thus, we define $\mathsf{cautious}_R(\exists R.A, \mathbf{B}_k, \mathcal{D}) = \langle v_A^R, \{A\}, \prec \rangle$, where $\prec$ is an arbitrary R-admissible ordering. A benefit of this strategy is that context v_A^R is reused only to satisfy existential restrictions of the form $\exists R.A$; thus, the literal ordering used in the context never needs to be refined.

The cautious and the eager strategies appear to be most natural, striking a different balance between the number of contexts and the number of inferences in each context. In Section 12.1 we investigate the effectiveness of these strategies in practice. Our results suggest that, in most practical cases, the eager strategy may be most appropriate because it does not introduce too many contexts; however, we also identified cases in which the cautious strategy is more appropriate. This suggests that a hybrid approach may be useful: one uses the eager strategy until some predetermined number of contexts is introduced, after which one switches to the cautious strategy.

Although the eager strategy can, in general, introduce an exponential number of contexts, as we argue next, on $\mathcal{EL}$ and DL-Lite$_{horn}$ ontologies the eager strategy introduces only linearly many contexts. First, if $\mathcal{O}$ is an $\mathcal{EL}$ ontology, then all existential restrictions in $\mathcal{O}$ contain only atomic roles, and all universal restrictions in $\mathcal{O}$ contain only inverse roles; hence, the Succ rule is applicable only with $\mathbf{B}_k = \emptyset$, and so the eager strategy coincides with the cautious strategy. Thus, similarly to the $\mathcal{EL}$ algorithm from Section 4.4, the eager strategy introduces only one context for each atomic concept. Second, if $\mathcal{O}$ is a DL-Lite$_{horn}$ ontology, then existential and universal restrictions occur in $\mathcal{O}$ only in axioms of the form $A \sqsubseteq \exists R.C_\top$ and $\top \sqsubseteq \forall R.B$ where $C_\top$ is a distinguished atomic concept; hence, if we apply the Hyper rule before the Succ rule, then all clauses $\top \sqsubseteq \forall R.A$ will be eagerly derived in each context, and so the Succ rule will be applicable to an existential restriction $\exists R.C_\top$ only with $\mathbf{B}_k = \{B \mid \top \sqsubseteq \forall R.B \in \mathcal{O}\}$; since this set is uniquely identified by R, the eager strategy will introduce at most one context for each role occurring in an existential restriction. This is similar to existing DL-Lite$_{horn}$ reasoning algorithms [5, 27]

8.6 Experimental Evaluation

The previous experiments in Chapter 6 suggest that, on $\mathcal{EL}$ ontologies, specialized consequence-based reasoners often outperform the more general-purpose tableau-based reasoners. Moreover, the good performance of consequence-based algorithms is known to extend to Horn-$\mathcal{SHIQ}$ ontologies [60]. The main goal of the evaluation presented in this section was to test whether this can also be said about non-Horn ontologies, and whether the implementation overhead of supporting disjunctions would impair the performance of a reasoner on Horn ontologies.

We have implemented the algorithm described in this chapter in a prototype reasoner ConDOR.[2] The reasoner currently implements only the $\mathsf{cautious}_R$ expansion strategy and, since at the time of the implementation our formulation of the algorithm did not include inverse roles [120], the reasoner only supports the DL $\mathcal{SH}$. Experiments with other expansion strategies and extensions of the reasoner to more expressive logics (we have already started to extend the consequence-based framework to $\mathcal{SHIQ}$ [119]) are left for future work.

We compared the performance of ConDOR r.12 with tableau-based reasoners FaCT++ 1.6.0 [132], HermiT 1.3.6 [94], and Pellet 2.3.0 [121], and the consequence-based Horn-$\mathcal{SHIQ}$ reasoner CB r.12 [60]. The experiments were executed on a laptop with Intel Core i7-2630QM 2GHz quad-core CPU and 6GB RAM running Microsoft Windows 7. We set a time-out of 1 hour and Java heap space to 4GB. We ran ConDOR using its command-line interface, CB as a plugin in Protégé 4.2, and we accessed the tableau-based reasoners through the OWL API 3.4. For all reasoners we only measured the classification time, which excludes parsing and loading.

Many existing ontologies were either created by translations from less expressive knowledge-representation formalisms, which do not support disjunctions, or designed directly in OWL and contain many other constructors. Consequently, there are very few ontologies that contain disjunctions but are still in $\mathcal{SH}$. One example of a large $\mathcal{SH}$ ontology with a significant number of disjunctions is the new SNOMED CT anatomical model mentioned in Section 1.2.2, which we call here SCT-SEP.[3] It contains 54,973 concepts, of which 18,323 are defined using disjunctions, and 9 roles, including one transitive role. Another prominent ontology with disjunctions is the NCI thesaurus, which is in the DL $\mathcal{SH}(\mathcal{D})$. We experimented with NCI version 12.04e from which we discarded all axioms

[2] `condor-reasoner.googlecode.com/`

[3] SCT-SEP is available at `http://condor-reasoner.googlecode.com/files/SCT-SEP.owl`.

	ConDOR	FaCT++	HermiT	Pellet	CB
NCI	4.0	37.2	68.6	184.2	N/A
SCT-SEP	84.2	1509.0	time	mem	N/A
GALEN-OWL	4.4	time	time	mem	3.7
SNOMED CT	43.8	425.2	time	mem	36.5

Table 8.2: Classification times in seconds

involving datatypes.[4] Finally, in order to evaluate the performance of ConDOR on Horn ontologies, we also included GALEN-OWL and SNOMED CT which were described already in Section 6.2.

The results of our experiments are shown in Table 8.2; possible failures for a reasoner are *time* (no result after 1 hour), *mem* (out-of-memory error), and *N/A* (the CB reasoner does not support non-Horn features). On the two Horn ontologies ConDOR shows similar improvement in performance over tableau-based reasoners as CB. Moreover, ConDOR retains this improvement even on non-Horn ontologies, reducing the classification time on SCT-SEP from over 25 minutes (for FaCT++) to under 2 minutes.

[4]NCI version 12.04e is available at `http://www.cs.ox.ac.uk/isg/ontologies/UID/00786.owl`.

Part IV

Parameterized Reasoning in DLs

Uproot your questions from their ground and the dangling roots will be seen. More questions!

—Frank Herbert
Chapterhouse: Dune

Chapter 9

Analyzing And-Branching Using ϵ-Free Decompositions

In the rest of this thesis we develop our framework for a parametric analysis of reasoning complexity. As we mentioned in the introduction, and-branching and or-branching are two main sources of reasoning complexity [12]. In this chapter we focus on quantifying the effects of and-branching, whereas in the next chapter we turn our attention to or-branching.

Our framework is based on the central notion of a *decomposition* $\mathcal{D}$ of an ontology $\mathcal{O}$ and a set of queries $\mathcal{Q}$. Intuitively, $\mathcal{D}$ is a graph-like structure that summarizes the models of $\mathcal{O}$ relevant for answering the queries in $\mathcal{Q}$; each vertex of $\mathcal{D}$ identifies a propositional subproblem that could occur in these models, and each edge of $\mathcal{D}$ identifies a pathway for the exchange of information between such subproblems. We then show how to instantiate our consequence-based framework from Part III so that it can be applied to $\mathcal{D}$, and we identify the *length* and *width* of $\mathcal{D}$ as parameters that characterize the and- and or-branching encountered in this process.

In Section 9.1 we first recapitulate several definitions from parameterized complexity [34]. We present an intuitive explanation of decompositions in Section 9.2, and we formalize them in Section 9.3. In Section 9.4 we give a practical algorithm for constructing decompositions. Finally, in Section 9.5 we show that decompositions also explain to an extent the difficulty of tableau reasoning: decomposition width bounds the size of labels annotating tableau vertices, so length and width bound the size of the constructed model representations.

9.1 Fixed-Parameter Tractable Problems

A *parameterized problem* is a set $P \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet and $\mathbb{N}$ is the set of nonnegative integers. Each pair $\langle x, k \rangle \in \Sigma^* \times \mathbb{N}$ is called a *problem instance*; x is called the *input*; k called is the *parameter*; and $||x||$ is the length of x. Problem P is *fixed-parameter tractable* (FPT) if a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant c, and an algorithm exist such that, given an arbitrary pair $\langle x, k \rangle \in \Sigma^* \times \mathbb{N}$, the algorithm decides $\langle x, k \rangle \in P$ in at most $f(k) \cdot ||x||^c$ steps. FPT is the class of all fixed-parameter tractable problems.

Some authors use an alternative definition of parameterized complexity. Just like a regular problem, a parameterized problem P is defined as a subset of Σ^*. Furthermore, a *parameterization* is a function $\kappa : \Sigma^* \to \mathbb{N}$. A problem P is fixed-parameter tractable w.r.t. a parameterization κ if a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant c, and an algorithm exist such that, for an arbitrary $x \in \Sigma^*$, the algorithm decides $x \in P$ in at most $f(\kappa(x)) \cdot ||x||^c$ steps. The two definitions are clearly equivalent if, for each $x \in \Sigma^*$, value $\kappa(x)$ is computable in polynomial time. Furthermore, even if the latter is not the case, the two definitions coincide if a computable function g and a constant d exist such that, for each fixed k, one can check whether $\kappa(x) \leq k$ holds using at most $g(k) \cdot ||x||^d$ steps; in other words, checking $\kappa(x) \leq k$ must be fixed-parameter tractable as well. The results we present in this paper involve the computation of a tree decomposition of a fixed width, which satisfies the latter condition.

9.2 Intuitions

Let $\mathcal{O}$ and q be as specified in Example 8.16, and let $\mathbf{L}$ be the set of all literals occurring in $\mathcal{O}$ and q; note that $|\mathbf{L}| = 13$. Proposition 8.13 provides us with an estimate of the complexity of deciding $\mathcal{O} \models q$: we can introduce at most $2^{|\mathbf{L}|} = 2^{13}$ contexts using the eager strategy, and each set $\mathcal{S}(v)$ can contain at most $4^{|\mathbf{L}|} = 4^{13}$ clauses, so the algorithm runs in time exponential in $|\mathbf{L}| = 13$. This, however, is a crude estimate because both the number of contexts and the sizes of sets $\mathcal{S}(v)$ are vastly overestimated.

To obtain a better estimate of these values, in this section we extend the notion of a context structure (cf. Definition 8.3) to a *decomposition* of an ontology and a set of queries; since decompo-

sitions are proper extensions of context structures, one can apply our consequence-based algorithm to a decomposition by only slightly adapting the initialization phase of the algorithm. Apart from $\exists R.A$-edges, a decomposition can also contain ϵ-edges, which we will use in Chapter 10 to analyze or-branching. In the rest of this chapter, however, we focus on and-branching and thus consider only *ϵ-free* decompositions.

An ϵ-free decomposition $\mathcal{D}$ associates with each context v three sets of literals: $\mathrm{core}(v)$, $\mathrm{knw}(v)$, and $\mathrm{poss}(v)$. These sets describe the clauses that the consequence-based algorithm can derive when applied to $\mathcal{O}$, $\mathcal{Q}$, and $\mathcal{D}$. Set $\mathrm{core}(v)$ serves the same purpose as in context structures—that is, the literals in $\mathrm{core}(v)$ are assumed to hold for all model elements corresponding to v. Furthermore, set $\mathrm{knw}(v)$ contains the *known* literals that are derived to hold for elements corresponding to v—that is, for each literal $L \in \mathrm{knw}(v)$, the algorithm will derive $\top \sqsubseteq L$ in $\mathcal{S}(v)$. Finally, set $\mathrm{poss}(v)$ contains the *possible* literals that might hold for elements corresponding to v—that is, each clause $K \sqsubseteq M \in \mathcal{S}(v)$ will satisfy $K \cup M \subseteq \mathrm{poss}(v)$. Clearly, all core literals are known, and all known literals are possible, so $\mathcal{D}$ should satisfy $\mathrm{core}(v) \subseteq \mathrm{knw}(v) \subseteq \mathrm{poss}(v)$ for each context v. To ensure that each clause obtained by applying the consequence-based algorithm to $\mathcal{D}$ is a logical consequence of $\mathcal{O}$, we require $\mathcal{D}$ to be *sound*: in addition to the condition on $\exists R.A$-edges from Definition 8.7, the known literals in each context of $\mathcal{D}$ must logically follow from the context's core—that is, $\mathcal{O} \models \mathrm{core}(v) \sqsubseteq L$ should hold for each context v in $\mathcal{D}$ and each literal $L \in \mathrm{knw}(v)$. Furthermore, for completeness, we require $\mathcal{D}$ to be *admissible*: roughly speaking, this notion requires $\mathcal{D}$ to contain all contexts that might be needed during our algorithm's execution, and sets $\mathrm{core}(v)$, $\mathrm{knw}(v)$, and $\mathrm{poss}(v)$ correctly describe the types of clauses derived by our algorithm. This will ensure that (i) the Succ rule is never applicable, and (ii) each clause in $\mathcal{S}(v)$ is of the form $\top \sqsubseteq L$ with $L \in \mathrm{knw}(v)$, or of the form $K \sqsubseteq M$ with $K \cup M \subseteq \mathrm{poss}(v) \setminus \mathrm{knw}(v)$. These observations allow us to capture the complexity of our algorithm using the following two parameters: decomposition *length* $\ln(\mathcal{D})$, defined as the number of contexts in $\mathcal{D}$, captures the combinatorial difficulty due to (i); and decomposition *width* $\mathrm{wd}(\mathcal{D})$, defined as the maximum cardinality of $\mathrm{poss}(v) \setminus \mathrm{knw}(v)$, captures the combinatorial difficulty due to (ii). We show that, when applied to $\mathcal{D}$, our consequence-based algorithm runs in time polynomial in $\ln(\mathcal{D})$ and $4^{\mathrm{wd}(\mathcal{D})^2}$; the latter factor is due to the fact that each set $\mathcal{S}(v)$ can contain at most $|\mathrm{knw}(v)| + 4^{|\mathrm{poss}(v) \setminus \mathrm{knw}(v)|}$ clauses.

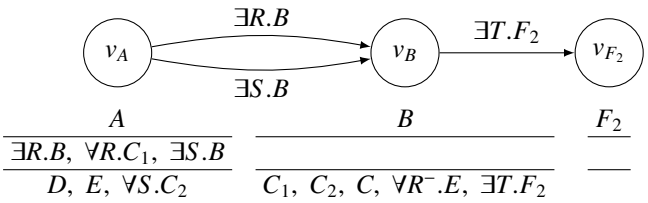
An ontology and a set of queries can admit infinitely many sound and admissible decomposi-

tions. Ideally, a decomposition $\mathcal{D}$ with least $\ln(\mathcal{D})$ and $\mathrm{wd}(\mathcal{D})$ would provide us with a "general" difficulty of reasoning with a particular ontology and a set of queries. This would be analogous to propositional satisfiability, where the treewidth of a propositional problem is defined as the smallest width over all tree decompositions of the problem. However, identifying such $\mathcal{D}$ is difficult for two reasons. First, checking semantic conditions in the definition of soundness requires reasoning, so verifying decomposition soundness can be as hard as the reasoning problem whose complexity we aim to analyze. Second, ontologies and queries exist for which all decompositions of minimal width have exponential length; we prove this claim for general (i.e., not only ϵ-free) decompositions, so we postpone the formal treatment of this property until Section 11.3. Hence, unlike for propositional satisfiability, minimal width can require exponential length, which essentially prevents us from obtaining a "general" characterization of reasoning difficulty.

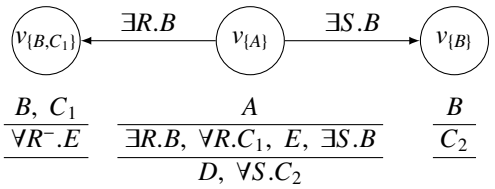
As a practical solution to these challenges, in Section 9.4 we present an algorithm for constructing some ϵ-free decomposition of $\mathcal{O}$ and $\mathcal{Q}$. To address the first issue, our algorithm approximates the entailment relation used in the definition of soundness. Thus, for some context v, set $\mathrm{knw}(v)$ approximates the set of literals entailed by $\mathrm{core}(v)$, and for each literal in $\mathrm{knw}(v)$ the relevant entailment can be established in polynomial time. To address the second issue, the algorithm is given a set of parameters $\mathcal{C}$ called a *control* that consists of an expansion strategy, an integer mln that imposes a bound on decomposition length, and two other parameters that determinize the decomposition construction process. For each $\mathcal{C}$, the ϵ-free decomposition of $\mathcal{O}$ and $\mathcal{Q}$ that is deterministically constructed by our algorithm is called the $\mathcal{C}$-*decomposition* of $\mathcal{O}$ and $\mathcal{Q}$. Parameter mln bounds the length of the $\mathcal{C}$-decomposition, which determines decomposition width. Thus, the width and length of a $\mathcal{C}$-decomposition provide us with an account of the reasoning difficulty relative to $\mathcal{C}$.

To illustrate these ideas, Figure 9.1 shows two ϵ-free decompositions of $\mathcal{O}$ and q from Example 8.16 obtained using the cautious and the eager expansion strategy. We use the same notation as for context structures, but show sets $\mathrm{core}(v)$, $\mathrm{knw}(v) \setminus \mathrm{core}(v)$, and $\mathrm{poss}(v) \setminus \mathrm{knw}(v)$ in a table below each context.

The ϵ-free decomposition $\mathcal{D}_1$ constructed from $\mathcal{O}$ and q using the cautious strategy is shown in Figure 9.1a. Since our goal is to check the entailment of $q = A \sqsubseteq E$, we introduce context v_A and add A to $\mathrm{core}(v_A)$, thus capturing the fact that the consequence-based algorithm initializes $\mathcal{S}(v_A)$ with (58). But then, from clauses (8.31), (8.32), and (8.34), the consequence-based algorithm can derive



(a) cautious strategy



(b) eager strategy

Figure 9.1: Examples of ϵ-free decompositions

(59), (60), and (66); to reflect this, we add literals $\exists R.B$, $\forall R.C_1$, and $\exists S.B$ to knw(v_A). Furthermore, using clause (8.35), the consequence-based algorithm will derive (67); however, the consequent of the clause contains a disjunction, so neither of the literals is derived deterministically, and therefore we add D and $\forall S.C_2$ to poss(v_A). Next, we must satisfy existential restrictions $\exists R.B$ and $\exists S.B$; since we use the cautious strategy, we introduce a single context v_B, add the two edges from v_A to v_B, and initialize core(v_B) to B; furthermore, due to universal restrictions $\forall R.C_1$ and $\forall S.C_2$ in poss(v_A), we add C_1 and C_2 to poss(v_B). Note that, due to the reuse of v_B, we cannot add C_1 to knw(v_B): it is not the case that all domain elements represented by v_B will necessarily satisfy C_1. We now continue our estimation for v_B: literal $\forall R^-.E$ is possible due to (8.33), concept C is possible due to (8.37), and literal $\exists T.F_2$ is possible due to (8.38). Moreover, since $\forall R^-.E \in$ poss(v_B), literal E is possible for v_A. Note that the consequence-based algorithm derives (65); however, $\forall R^-.E$ is not in knw(v_B) due to context reuse, so we do not have sufficient information to add E to knw(v_A); in other words, knw(v_A) provides us only with a lower bound on "truly" known literals. Finally, we introduce context v_{F_2} to satisfy $\exists T.F_2 \in$ poss(v_B) and thus obtain an admissible ϵ-free decomposition. Clearly, wd($\mathcal{D}_1$) = 5 and ln($\mathcal{D}_1$) = 3 so, when applied to $\mathcal{D}_1$, our inference rules can derive $3 \cdot 4^5$ clauses, which is a much more accurate estimate.

The ϵ-free decomposition $\mathcal{D}_2$ constructed from $\mathcal{O}$ and q using the eager strategy is shown in Figure 9.1b. The main difference compared to the previous case is that, due to the eager strategy,

we now use distinct contexts, $v_{\{B,C_1\}}$ and $v_{\{B\}}$, to satisfy existential restrictions $\exists R.B$ and $\exists S.B$ at context $v_{\{A\}}$. Since literal $\forall R.C_1$ is known at $v_{\{A\}}$, we can now add C_1 to the core of $v_{\{B,C_1\}}$, which in turn allows us to make E known at $v_{\{A\}}$. Furthermore, literal $\exists T.F_2$ is now not possible at $v_{\{B\}}$. In this case, we have $\mathrm{wd}(\mathcal{D}_2) = 2$ and $\ln(\mathcal{D}_1) = 3$ so, when applied to $\mathcal{D}_2$, our inference rules can derive at most $3 \cdot 4^2$ clauses; thus, we have obtained an even better estimate of the running time of our consequence-based algorithm.

Reasoning with Horn-$\mathcal{ALCI}$ ontologies is generally easier in practice: such ontologies admit a single canonical model, which eliminates all or-branching; however, due to and-branching, reasoning is still ExpTime-hard [74]. This is reflected in our notions of decompositions. In particular, if control $\mathcal{C}$ uses the eager expansion strategy and allows for "sufficiently many" contexts, the $\mathcal{C}$-decomposition $\mathcal{D}$ of a Horn ontology $\mathcal{O}$ has zero width. Then, all possible literals in $\mathcal{D}$ are also known, no or-branching is required, and so $\mathcal{D}$ contains a complete solution to the reasoning problem. By reducing the maximum number of contexts in $\mathcal{C}$, however, we can "trade off" and-branching for or-branching. Then, $\mathcal{D}$ does not necessarily have zero width, so further reasoning is needed; moreover, when applied to $\mathcal{D}$, our consequence-based algorithm derives only Horn clauses, and the size of the antecedent of each clause is bounded by $\mathrm{wd}(\mathcal{D})$. Furthermore, as we discussed in Section 8.5.2, if $\mathcal{O}$ is an $\mathcal{EL}$ or a DL-Lite$_{horn}$ ontology, the eager strategy introduces only linearly many contexts. Therefore, if $\mathcal{C}$ uses the eager strategy, the $\mathcal{C}$-decomposition of $\mathcal{O}$ has zero width and linear length, thus explaining the tractability of subsumption reasoning in $\mathcal{EL}$ and DL-Lite$_{horn}$.

9.3 Formalization

Throughout this section we fix a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$ and a finite set of queries $\mathcal{Q}$; furthermore, we let $\mathbf{L}$ be the set of literals that occur in $\mathcal{O} \cup \mathcal{Q}$, by $\mathbf{L}^{\exists}$ we denote the set of all existential restrictions from $\mathbf{L}$, and we let ϵ be a special symbol not occurring in $\mathbf{L}$. Please recall that $\mathbf{X}$ is the countably infinite set of all contexts that was fixed in Section 8.3. We next formalize the notion of decompositions and then generalize the notion of context structure soundness to decompositions. In order to simplify the presentation, Definitions 9.1 and 9.2 both consider ϵ-edges that are used only in Chapter 10 for the analysis of or-branching; however, in this section we subsequently consider only decompositions without ϵ-edges.

Definition 9.1. A *decomposition* of $\mathcal{O}$ and $\mathcal{Q}$ is a tuple $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \text{knw}, \text{poss}, \prec, \vartheta \rangle$ where $\mathcal{V} \subseteq \mathbf{X}$ is a finite set of contexts, set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times (\mathbf{L}^{\exists} \cup \{\epsilon\})$ contains edges labeled by existential restrictions (an $\exists R.A$-*edge*) or by ϵ (an ϵ-*edge*), functions $\text{core}, \text{knw}, \text{poss} \colon \mathcal{V} \to 2^{\mathbf{L}}$ label contexts with sets of literals, function $\prec$ assigns to each context $v \in \mathcal{V}$ a literal ordering $\prec_v$, and function $\vartheta \colon \mathcal{Q} \to \mathcal{V}$ maps queries to contexts. For each vertex $v \in \mathcal{V}$, let $\text{uknw}(v) \coloneqq \text{poss}(v) \setminus \text{knw}(v)$. The *length* of $\mathcal{D}$ is $\text{ln}(\mathcal{D}) \coloneqq |\mathcal{V}|$, and the *width* of $\mathcal{D}$ is $\text{wd}(\mathcal{D}) \coloneqq \max_{v \in \mathcal{V}} |\text{uknw}(v)|$. Decomposition $\mathcal{D}$ is ϵ-*free* if it contains no ϵ-edges.

Definition 9.2. A decomposition $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \text{knw}, \text{poss}, \prec, \vartheta \rangle$ of $\mathcal{O}$ and $\mathcal{Q}$ is *sound* if it satisfies all of the following conditions.

(N1) $\mathcal{O} \models \text{core}(v) \sqcap \exists R.A \sqsubseteq \exists R.[\text{core}(u) \sqcap A]$ for each $\exists R.A$-edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$.

(N2) $\text{core}(v) = \text{core}(u)$ for each ϵ-edge $\langle v, u, \epsilon \rangle \in \mathcal{E}$.

(N3) $\mathcal{O} \models \text{core}(v) \sqsubseteq L$ for each context $v \in \mathcal{V}$ and each $L \in \text{knw}(v)$.

Condition (N1) of Definition 9.2 is the same as in Definition 8.7; condition (N2) is relevant only for decompositions that are not ϵ-free and will be explained in Chapter 10; and condition (N3) says that known literals should hold in all domain elements represented by a specific context. Please note that condition (N3) imposes only an upper bound on the known literals; thus, $\text{knw}(v)$ can be obtained using a sound but incomplete technique such as the one we present in Section 9.4.

Next, we extend the notion of context structure admissibility to decompositions. The ordering condition is the same as in context structures. Furthermore, to encapsulate all relevant parameters for the consequence-based algorithm, a decomposition contains a covering mapping, and the covering condition restates Definition 8.9. The ontology condition says that, if $\mathcal{O}$ contains a clause $K \sqsubseteq M$ and all of the literals in K are possible at context v, then the Hyper rule can derive M and so all literals in M should be possible at v too. Finally, the structural condition (S1) captures the intuitive relationship between the core, known, and possible literals; condition (S2) ensures that a decomposition correctly estimates all consequences of the Pred rule; and condition (S3) ensures that a decomposition contains sufficiently many contexts so that the Succ rule is never applicable.

Definition 9.3. Let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \text{knw}, \text{poss}, \prec, \vartheta \rangle$ be an ϵ-free decomposition of $\mathcal{O}$ and $\mathcal{Q}$. Let $v \in \mathcal{V}$ be an arbitrary context of $\mathcal{D}$, and let $K \sqsubseteq M$ be an arbitrary query; then, v is *sound* for $K \sqsubseteq M$

if $\text{core}(v) \subseteq K$; v is *complete* for $K \sqsubseteq M$ if $K \subseteq \text{poss}(v)$ and M is $\prec_v$-minimal; and v *covers* $K \sqsubseteq M$ if v is both sound and complete for $K \sqsubseteq M$. Finally, $\mathcal{D}$ is *admissible* if all of the following conditions are satisfied.

- Structural conditions:

  (S1) For each context $v \in \mathcal{V}$, we have $\text{core}(v) \subseteq \text{knw}(v) \subseteq \text{poss}(v)$.

  (S2) For each edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ and each $\forall \mathsf{inv}(R).C \in \text{poss}(u)$, we have $C \in \text{poss}(v)$.

  (S3) For each literal $\exists R.A \in \text{poss}(v)$, there exists an edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ such that

  $$\text{poss}(u) \supseteq \{A\} \cup \{B \mid \forall R.B \in \text{poss}(v)\}.$$

- Ordering condition: For each edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$, the literal ordering $\prec_u$ is R-admissible.

- Ontology condition: For each context $v \in \mathcal{V}$ and each clause $K \sqsubseteq M \in \mathcal{O}$ with $K \subseteq \text{poss}(v)$, we have $M \subseteq \text{poss}(v)$.

- Covering condition: Each query $q \in \mathcal{Q}$ is covered in the context $\vartheta(q)$.

We can apply the consequence-based algorithm to a sound, admissible, and ϵ-free decomposition $\mathcal{D}$ as specified in Algorithm 9.4. Step 1 initializes each $\mathcal{S}(v)$ with a clause $\top \sqsubseteq L$ for each known literal $L \in \text{knw}(v)$. This allows us to prove in Lemma 9.5 that the algorithm derives only clauses of the form $K \sqsubseteq M$ with $K \cup M \subseteq \text{uknw}(v)$, which is central to our complexity argument in Proposition 9.7. Please note that, if step 1 were to consider only the literals in $\text{core}(v)$, our algorithm would eventually derive a clause $\top \sqsubseteq L$ for each known literal $L \in \text{knw}(v)$, but this might involve clauses containing combinations of literals in $\text{knw}(v)$; thus, step 1 introduces derivation "shortcuts" for the known literals, which can improve the algorithm's performance in practice. Step 2 is analogous to the way the Succ rule initializes contexts, and it ensures that each atomic concept B that can be propagated to a context u through existential or universal quantification is relevant for u. Step 3 ensures that each query $q \in \mathcal{Q}$ is covered in context $\vartheta(q)$. Finally, step 4 applies exhaustively the consequence-based inference rules, but without the Succ rule. The latter is possible because $\mathcal{D}$ is admissible and thus contains all relevant contexts, which allows us to prove in Lemma 9.6 that the

Succ rule is never applicable during the algorithm's execution. By combining this observation with the fact that $\mathcal{D}$ is sound, in Proposition 9.7 we show that our algorithm is sound and complete.

Algorithm 9.4. The *ϵ-free decomposition algorithm* for $\mathcal{ALCI}$ takes as its argument a sound, admissible, and ϵ-free decomposition $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \text{knw}, \text{poss}, \prec, \vartheta \rangle$ of $\mathcal{O}$ and $\mathcal{Q}$. The algorithm (nondeterministically) computes a clause system $\mathcal{S}$ for $\mathcal{D}$ as follows.

1. Set $\mathcal{S}(v) \coloneqq \{\top \sqsubseteq L \mid L \in \text{knw}(v)\}$ for each context $v \in \mathcal{V}$.
2. For each edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ and each concept $B \in \text{uknw}(u)$ such that $B = A$ or $\forall R.B \in \text{poss}(v)$, add $B \sqsubseteq B$ to $\mathcal{S}(u)$.
3. For each query $K \sqsubseteq M \in \mathcal{Q}$, let $v \coloneqq \vartheta(K \sqsubseteq M)$; then, for each literal $L \in K \setminus \text{knw}(u)$, add $L \sqsubseteq L$ to $\mathcal{S}(v)$.
4. Exhaustively apply rules Hyper, Pred, and Elim from Table 8.1.

Lemma 9.5. *Let $\mathcal{S}$ be an arbitrary clause system encountered during step 4 of Algorithm 9.4, and let $v \in \mathcal{V}$ be an arbitrary context in $\mathcal{D}$. Then, each clause in $\mathcal{S}(v)$ is of the form $\top \sqsubseteq L$ with $L \in \text{knw}(v)$, or $K \sqsubseteq M$ with $K \cup M \subseteq \text{uknw}(v)$.*

Proof. Let $\mathcal{S}$ be a clause system at the beginning of step 4, and let $v \in \mathcal{V}$ be an arbitrary context. By Definition 9.3, if context v covers some query $K \sqsubseteq M$, then $K \subseteq \text{poss}(v)$ holds; thus, each clause added to $\mathcal{S}(v)$ in steps 1–3 is clearly of the required form. We next show that this property is preserved in step 4.

Assume that the Hyper rule is applicable to some clause $\bigsqcap_{i=1}^{n} A_i \sqsubseteq M \in \mathcal{O}$. By the induction assumption, each premise from $\mathcal{S}(v)$ is of the form $K_i \sqsubseteq M_i \sqcup A_i$ with $K_i \cup M_i \subseteq \text{uknw}(v)$ and $A_i \in \text{poss}(v)$. Thus, $\sqcap_{i=1}^{n} A_i \subseteq \text{poss}(v)$, so $M \subseteq \text{poss}(v)$ by the ontology condition of Definition 9.3. The Hyper rule produces the clause $\bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i$. Now assume that there exists a literal $L \in M \cap \text{knw}(v)$; then, $\top \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ due to step 1 of Algorithm 9.4 and Lemma 8.11, which contradicts the assumption that the Hyper rule is applicable. Hence, $M \subseteq \text{uknw}(v)$, and the produced clause is of the required form.

Assume that the Pred rule is applicable to an edge $\langle v, u, \exists R.A \rangle, \in \mathcal{E}$. By the induction assumption, the premises are of the form $K_0 \sqsubseteq M_0 \sqcup \exists R.A$ and $K_i \sqsubseteq M_i \sqcup \forall R.B_i$, where $K_i \cup M_i \subseteq \text{uknw}(v)$

for each $0 \leq i \leq n$. The Pred rule produces the clause $\bigsqcap_{i=0}^{n} K_i \sqsubseteq \bigsqcup_{i=0}^{n} M_i \sqcup \bigsqcup_{j=1}^{m} C_j$. Now for each $1 \leq j \leq m$, by the induction assumption we have $\forall \mathsf{inv}(R).C_j \in \mathrm{poss}(u)$, so $C_j \in \mathrm{poss}(v)$ by condition (S2) of Definition 9.3. Furthermore, if $C_j \in \mathrm{knw}(v)$, then we have $\top \sqsubseteq C_j \mathrel{\hat{\in}} \mathcal{S}(v)$ due to step 1 of Algorithm 9.4 and Lemma 8.11, which contradicts the assumption that the Pred rule is applicable; consequently, we have $C_j \in \mathrm{uknw}(v)$. Hence, the clause produced by the rule is of the required form. □

Lemma 9.6. *The* Succ *rule is never applicable during step 4 of Algorithm 9.4.*

Proof. Let $\mathcal{S}$ be a clause system encountered in step 4 of Algorithm 9.4; let $v \in \mathcal{V}$ be a context; let $K \sqsubseteq M \sqcup \exists R.A \in \mathcal{S}(v)$ be a clause; and let $\mathbf{B}_p$ be as specified in Table 8.1. By Lemma 9.5, we have $\{B \mid \forall R.B \in \mathrm{poss}(v)\} \supseteq \mathbf{B}_p$ and $\exists R.A \in \mathrm{poss}(v)$. Thus, by condition (S3) of Definition 9.3, an edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ exists with $\mathrm{poss}(u) \supseteq \{A\} \cup \{B \mid \forall R.B \in \mathrm{poss}(v)\} \supseteq \{A\} \cup \mathbf{B}_p$; furthermore, due to steps 1 and 2 of Algorithm 9.4 and Lemma 8.11, we have that $L \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(u)$ holds for each $L \in \{A\} \cup \mathbf{B}_p$. Thus, the Succ rule is not applicable to context v and clause $K \sqsubseteq M \sqcup \exists R.A$ in $\mathcal{S}(v)$. □

Proposition 9.7. *Let $\mathcal{S}$ be a clause system obtained by applying Algorithm 9.4 to $\mathcal{O}$, $\mathcal{Q}$, and $\mathcal{D}$. Then, for each query $q \in \mathcal{Q}$, we have $\mathcal{O} \models q$ if and only if $q \mathrel{\hat{\in}} \mathcal{S}(\vartheta(q))$.*

Proof. Consider an arbitrary query $q = K \sqsubseteq M \mathrel{\hat{\in}} \mathcal{Q}$ and let $v = \vartheta(q)$; context v covers q by the covering condition of Definition 9.3, Decomposition $\mathcal{D}$ is sound, so, by condition (N3) of Definition 9.2, we have $\mathcal{O} \models \mathrm{core}(u) \sqsubseteq L$ for each clause $\top \sqsubseteq L$ added to some $\mathcal{S}(u)$ in step 1 of Algorithm 9.4; furthermore, clauses added in step 2 and 3 are tautologies; consequently, $\mathcal{S}$ is sound for $\mathcal{O}$. But then, since context v is sound for q, we have that $K \sqsubseteq M \mathrel{\hat{\in}} \mathcal{S}(v)$ implies $\mathcal{O} \models K \sqsubseteq M$ in the same way as in the proof of Proposition 8.12. Furthermore, for an arbitrary literal $L \in K$, either $\top \sqsubseteq L$ is added to $\mathcal{S}(v)$ in step 1, or $L \sqsubseteq L$ is added to $\mathcal{S}(v)$ in step 3; either way, we have $K \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$, and this property is preserved during the algorithm's execution by Lemma 8.11. Finally, Lemma 9.6 ensures that $\mathcal{S}$ is closed under the Hyper, Pred, and Succ rules; but then, since context v is complete for q, we have that $\mathcal{O} \models K \sqsubseteq M$ implies $K \sqsubseteq M \mathrel{\hat{\in}} \mathcal{S}(v)$ by Theorem 8.5. □

Proposition 9.8 provides us with our ultimate goal in this section: it shows that decomposition width and length provide us with a more accurate estimate of the complexity of subsumption

reasoning.

Proposition 9.8 (Termination)**.** *When applied to some $\mathcal{O}$, $\mathcal{Q}$, and a sound, admissible, and ϵ-free decomposition $\mathcal{D}$ of $\mathcal{O}$ and $\mathcal{Q}$, Algorithm 9.4 terminates in time polynomial in $4^{\mathrm{wd}(\mathcal{D})^2}$, $\ln(\mathcal{D})$, and $\|\mathcal{O}\| + \|\mathcal{Q}\|$.*

Proof. Let $d = \mathrm{wd}(\mathcal{D})$ and $m = \ln(\mathcal{D})$, and recall that $|\mathbf{L}| \leq \|\mathcal{O}\| + \|\mathcal{Q}\|$. We next show that the number of inferences is polynomial in 4^{d^2}, m, and $\|\mathcal{O}\| + \|\mathcal{Q}\|$; this observation implies the claim of this proposition in the same way as in the proof of Proposition 8.13. By Lemma 9.5, for each context v, each clause in $\mathcal{S}(v)$ is of the form

- $\top \sqsubseteq L$ with $L \in \mathrm{knw}(v)$, and set $\mathcal{S}(v)$ can contain at most $|\mathbf{L}|$ such clauses, or
- $K \sqsubseteq M$ with $K \cup M \subseteq \mathrm{uknw}(v)$, and set $\mathcal{S}(v)$ can contain at most $c = 4^d$ such clauses.

For the Hyper rule, a context $v \in \mathcal{V}$ can be chosen in at most m ways, and a clause $\bigsqcap_i A_i \sqsubseteq M \in \mathcal{O}$ can be chosen in at most $|\mathcal{O}|$ ways. For each $A_i \in \mathrm{uknw}(v)$ there are at most c ways of choosing a matching clause $K_i \sqsubseteq M_i \sqcup A_i \in \mathcal{S}(v)$, and for each $A_i \in \mathrm{knw}(v)$ there is exactly one way of choosing a matching clause $\top \sqsubseteq A_i \in \mathcal{S}(v)$. Consequently, there are at most c^d ways of choosing premises $K_i \sqsubseteq M_i \sqcup A_i \in \mathcal{S}(v)$.

For the Pred rule, an edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ can be chosen in at most $m^2 \cdot |\mathbf{L}|$ ways, and a clause $A \sqcap \bigsqcap_i B_i \sqsubseteq \bigsqcup_j \forall \mathsf{inv}(R).C_j \in \mathcal{S}(u)$ or $\bigsqcap_i B_i \sqsubseteq \bigsqcup_j \forall \mathsf{inv}(R).C_j \in \mathcal{S}(u)$ can be chosen in at most $|\mathbf{L}| + 4^d$ ways. Analogously to the Hyper rule, there are at most c^d ways of choosing premises $K_0 \sqsubseteq M_0 \sqcup \exists R.A \in \mathcal{S}(v)$ and $K_i \sqsubseteq M_i \sqcup \forall R.A \in \mathcal{S}(v)$. □

We would like to point out that the factor $4^{\mathrm{wd}(\mathcal{D})^2}$ in Proposition 9.8 is due to the fact that the Hyper and the Pred rules are based on hyperresolution: they fix one of the $4^{\mathrm{wd}(\mathcal{D})}$ clauses, and then try to find matches for each of the $\mathrm{wd}(\mathcal{D})$ literals in the antecedent, which gives rise to $4^{\mathrm{wd}(\mathcal{D})^2}$ combinations. It is, however, possible to develop a binary version of these inference rules, in which case the algorithm's complexity would be polynomial in $4^{\mathrm{wd}(\mathcal{D})}$. In our experience, hyperresolution tends to be more effective than binary resolution in practice, which is why we opted for this style of presentation.

9.4 Constructing ϵ-Free Decompositions

We now present our algorithm for computing ϵ-free decompositions that we outlined in Section 9.2. We first formalize the notion of a control, which encapsulates the parameters that guide our algorithm in the construction process. Similarly to Algorithm 8.10, these include initialization and expansion strategies, but also further parameters that ensure that the output of the algorithm is uniquely determined.

Definition 9.9. A *control* is a tuple $\mathcal{C} = \langle \text{mln}, \mathsf{initialization}, \mathsf{strategy}, \mathsf{precedence} \rangle$ with the following components.

- mln is a positive integer.
- $\mathsf{initialization}$ is a polynomial-time computable function taking a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$, a finite set of queries $\mathcal{Q}$, and a positive integer mln. The result of $\mathsf{initialization}(\mathcal{O}, \mathcal{Q}, \text{mln})$ is a pair $\langle \mathcal{D}, \vartheta \rangle$ where
  - $\mathcal{D}$ is a context structure over the literals occurring in $\mathcal{O} \cup \mathcal{Q}$ such that $\mathcal{D}$ contains no edges, $\ln(\mathcal{D}) \leq \text{mln}$, and $\mathcal{D}$ contains a trivial context, and
  - ϑ is a covering mapping from $\mathcal{Q}$ to $\mathcal{D}$.
- $\mathsf{strategy}$ is a polynomial-time computable function taking an existential restriction $\exists R.A$, a set of atomic concepts $\mathbf{B}_k$, a decomposition $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \text{knw}, \text{poss}, \prec, \vartheta \rangle$ that contains a trivial context, and a positive integer mln. The result of $\mathsf{strategy}(\exists R.A, \mathbf{B}_k, \mathcal{D}, \text{mln})$ is a triple $\langle u, \text{core}', \prec' \rangle$ where
  - core' is a subset of $\{A\} \cup \mathbf{B}_k$,
  - $\prec'$ is an R-admissible literal ordering, and
  - either $u \in \mathbf{X} \setminus \mathcal{V}$ is a fresh context, or $u \in \mathcal{V}$ is a context in $\mathcal{D}$ such that $\text{core}(u) = \text{core}'$.

  If $|\mathcal{V}| \geq \text{mln}$, then $\mathsf{strategy}$ must return a context $u \in \mathcal{V}$ from $\mathcal{D}$; since $\mathcal{D}$ contains a trivial context, a context satisfying the required conditions is guaranteed to exist.
- $\mathsf{precedence}$ is a strict total order on the set of all pairs of the form $\langle \exists R.A, v \rangle$ with $\exists R.A \in \Sigma_L^\exists$ and $v \in \mathbf{X}$.

$\mathbf{H}_k$	If	$K \subseteq \mathrm{knw}(v)$, $K \sqsubseteq L \in \mathcal{O}$, and $L \notin \mathrm{knw}(v)$,
	then	add L to $\mathrm{knw}(v)$.
$\mathbf{H}_p$	If	$K \subseteq \mathrm{poss}(v)$, $K \sqsubseteq M \in \mathcal{O}$, and $M \not\subseteq \mathrm{poss}(v)$,
	then	add each literal in M to $\mathrm{poss}(v)$.
$\mathbf{P}_k$	If	$\langle v, u, \exists R.A\rangle \in \mathcal{E}$, $\exists R.A \in \mathrm{knw}(v)$, $\forall \mathsf{inv}(R).C \in \mathrm{knw}(u)$, and $C \notin \mathrm{knw}(v)$,
	then	add C to $\mathrm{knw}(v)$.
$\mathbf{P}_p$	If	$\langle v, u, \exists R.A\rangle \in \mathcal{E}$, $\forall \mathsf{inv}(R).C \in \mathrm{poss}(u)$, and $C \notin \mathrm{poss}(v)$,
	then	add C to $\mathrm{poss}(v)$.
$\mathbf{S}$	If	$\exists R.A \in \mathrm{poss}(v)$, and for $\mathbf{B}_p = \{B \mid \forall R.B \in \mathrm{poss}(v)\}$, no edge $\langle v, u, \exists R.A\rangle \in \mathcal{E}$ exists such that $\mathrm{poss}(u) \supseteq \{A\} \cup \mathbf{B}_p$,
	then	let $\langle u, \mathrm{core}', \prec'\rangle \coloneqq \mathsf{strategy}(\exists R.A, \mathbf{B}_k, \mathcal{D}, \mathrm{mln})$ where $\mathbf{B}_k = \{B \mid \forall R.B \in \mathrm{knw}(v)\}$; if $u \in \mathcal{V}$, then let $\prec_u \coloneqq \prec_u \cap \prec'$, and otherwise let $\mathcal{V} \coloneqq \mathcal{V} \cup \{u\}$, $\mathrm{core}(u) \coloneqq \mathrm{knw}(u) \coloneqq \mathrm{poss}(u) \coloneqq \mathrm{core}'$, and $\prec_u \coloneqq \prec'$; add $\langle v, u, \exists R.A\rangle$ to $\mathcal{E}$; and add each literal in $\{A\} \cup \mathbf{B}_p$ to $\mathrm{poss}(u)$.

Table 9.1: ϵ-free decomposition construction rules

Intuitively, a control $\mathcal{C}$ specifies which contexts to introduce for a given ontology and a set of queries. In particular, function initialization specifies how to assign the queries to contexts; the function returns a context structure and a covering mapping that our algorithm will use as a starting point for the construction of an ϵ-free decomposition: the algorithm will immediately extend the given context structure into an ϵ-free decomposition by setting $\mathrm{poss}(v) \coloneqq \mathrm{knw}(v) \coloneqq \mathrm{core}(v)$ for each context v. Furthermore, function strategy is analogous to an expansion strategy (cf. Definition 8.6) and it specifies which contexts to introduce in order to satisfy existential and universal restrictions. Moreover, as we discussed in Section 9.2, the width and the length of a decomposition are not independent, and an exponential length may be needed to minimize the width. To overcome this problem, mln imposes an upper bound on decomposition length, and initialization and strategy are both required to honor this restriction. In particular, strategy is required to reuse a context whenever this bound has been exceeded; the decomposition being constructed will always contain the trivial context, so context reuse will always be possible. Apart from this modification, concrete instances of initialization and strategy are analogous to the concrete initialization and expansions strategies discussed in Section 8.5. Finally, precedence determines a unique order in which fresh contexts are introduced, so that the resulting ϵ-free decomposition is uniquely determined by $\mathcal{C}$, the ontology, and the set of queries.

Throughout the rest of section we fix a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$, a finite set of queries $\mathcal{Q}$, and a control $\mathcal{C}$; furthermore, we let $\mathbf{L}$ be the set of literals that occur in $\mathcal{O} \cup \mathcal{Q}$. We are now ready to present our decomposition construction algorithm.

Algorithm 9.10. The *ϵ-free decomposition construction algorithm* takes a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$, a finite set of queries $\mathcal{Q}$, and a control $\mathcal{C} = \langle \text{mln}, \mathsf{initialization}, \mathsf{strategy}, \mathsf{precedence} \rangle$, and it constructs an ϵ-free decomposition $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \text{knw}, \text{poss}, \prec, \vartheta \rangle$ of $\mathcal{O}$ and $\mathcal{Q}$ as follows.

1. Let $\langle \langle \mathcal{V}, \mathcal{E}, \text{core}, \prec \rangle, \vartheta \rangle \coloneqq \mathsf{initialization}(\mathcal{O}, \mathcal{Q}, \text{mln})$, and then set $\text{poss}(v) \coloneqq \text{knw}(v) \coloneqq \text{core}(v)$ for each context $v \in \mathcal{V}$.

2. For each query $K \sqsubseteq M \in \mathcal{Q}$, let $v \coloneqq \vartheta(K \sqsubseteq M)$; then, set $\text{poss}(v) \coloneqq \text{poss}(v) \cup K$.

3. Exhaustively apply all rules from Table 9.1 apart from rule **S**.

4. Stop and return $\mathcal{D}$ if rule **S** is not applicable.

5. Let $\langle \exists R.A, v \rangle$ be the minimal pair w.r.t. $\mathsf{precedence}$ such that rule **S** is applicable to $\exists R.A$ and v; then, apply rule **S** to $\exists R.A$ and v, and proceed to step 3.

A decomposition produced by this algorithm is called a *$\mathcal{C}$-decomposition* of $\mathcal{O}$ and $\mathcal{Q}$.

Algorithm 9.10 can be intuitively understood as follows. Step 1 uses function $\mathsf{initialization}$ to introduce the contexts needed to cover the queries in $\mathcal{Q}$, and then it initializes the sets of known and possible literals for each context to be equal to the context's core. Next, for each query $q = K \sqsubseteq M \in \mathcal{Q}$ and each literal $L \in K$, step 2 of Algorithm 9.10 adds L to $\text{poss}(\vartheta(q))$ in order to reflect the fact that step 3 of Algorithm 9.4 introduces a clause of the form $L \sqsubseteq L$ for each such L. Finally, in steps 3–5, the algorithm applies the rules shown in Table 9.1. Rules $\mathbf{H}_k$ and $\mathbf{H}_p$ correspond to the Hyper rule; rules $\mathbf{P}_k$ and $\mathbf{P}_p$ correspond to the Pred rule; and rule **S** corresponds to the Succ rule. Rules $\mathbf{H}_k$ and $\mathbf{P}_k$, however, are applied to the sets of known literals, and so they determine a lower bound on the known consequences of rules Hyper and Pred, respectively. In contrast, rules $\mathbf{H}_p$ and $\mathbf{P}_p$ are applied to the sets of possible literals, and so they determine an upper bound on the possible consequences of rules Hyper and Pred, respectively. Finally, rule **S** introduces all possible contexts that may be introduced by the Succ rule; the rule is applied with the lowest priority and in accordance with the precedence $\mathsf{precedence}$ from $\mathcal{C}$ in order to make the algorithm deterministic. In

the rest of this section we prove the following theorem, which summarizes the formal properties of Algorithm 9.10.

Theorem 9.11. *When applied to $\mathcal{O}$, $\mathcal{Q}$, and $\mathcal{C}$, Algorithm 9.10 terminates in time polynomial in* mln *and* $\|\mathcal{O}\| + \|\mathcal{Q}\|$. *The result is a unique, sound, admissible, and ϵ-free decomposition $\mathcal{D}$ of $\mathcal{O}$ and $\mathcal{Q}$ satisfying* $\ln(\mathcal{D}) \leq \text{mln}$.

Let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \text{knw}, \text{poss}, \prec, \vartheta \rangle$ be a decomposition obtained by the decomposition construction algorithm. Clearly, $\mathcal{D}$ refers only to the literals occurring in $\mathcal{O} \cup \mathcal{Q}$, and it does not contain ϵ-edges; hence, $\mathcal{D}$ is an ϵ-free decomposition of $\mathcal{O}$ and $\mathcal{Q}$. We next prove the remaining claims of Theorem 9.11.

Claim 9.12. *The decomposition construction algorithm terminates in time polynomial in* mln *and* $\|\mathcal{O}\| + \|\mathcal{Q}\|$, *and the resulting decomposition $\mathcal{D}$ satisfies* $\ln(\mathcal{D}) \leq \text{mln}$.

Proof. Since initialization is computable in polynomial time, steps 1 and 2 can be performed in time polynomial in mln and $\|\mathcal{O}\| + \|\mathcal{Q}\|$. By Definition 9.9, step 1 introduces at most mln contexts due to the requirement on initialization, and rule **S** can introduce a new context in the call to strategy only if the total number of contexts is smaller than mln; thus, $|\mathcal{V}| \leq \text{mln}$ clearly holds. Since $\mathcal{D}$ refers only to literals from $\mathbf{L}$, we have $|\mathcal{E}| \leq \text{mln}^2 \cdot |\mathbf{L}|$ and, for each context $v \in \mathcal{V}$, we have

$$|\text{core}(v)| \leq |\mathbf{L}|, \quad |\text{knw}(v)| \leq |\mathbf{L}|, \quad \text{and} \quad |\text{poss}(v)| \leq |\mathbf{L}|.$$

The decomposition construction algorithm is monotonic—that is, it never removes elements from the decomposition being constructed. Also, the precondition of each rule contains a negation of the rule's postcondition; thus, whenever a rule is applicable, its application actually modifies the decomposition.

Now let $m_{\mathcal{D}} = \text{mln}^2 \cdot |\mathbf{L}| + \text{mln} \cdot 3 \cdot |\mathbf{L}|$. The decomposition construction algorithm can modify the decomposition at most $m_{\mathcal{D}}$ times: it can add each of the $\text{mln}^2 \cdot |\mathbf{L}|$ possible edges, and it can add $|\mathbf{L}|$ concepts to core(v), knw(v), and poss(v) for each context $v \in \mathcal{V}$. Thus, after at most $m_{\mathcal{D}}$ modifications, no rule can further modify the decomposition. Therefore, since each rule application modifies the decomposition, the decomposition construction algorithm terminates after at most $m_{\mathcal{D}}$ rule applications. Furthermore, due to the fact that each rule precondition refers to a bounded

number of vertices, the preconditions of each rule instance can be checked in polynomial time. Since strategy is computable in polynomial time, each rule can be applied in polynomial time as well.

Thus, the algorithm requires a polynomial number of modifications of the decomposition, each of which requires polynomial time. Since $|\mathbf{L}| < \|\mathcal{O}\| + \|\mathcal{Q}\|$, the algorithm can be implemented so that it runs in time polynomial in mln and $\|\mathcal{O}\| + \|\mathcal{Q}\|$. □

Claim 9.13. *Decomposition $\mathcal{D}$ is unique.*

Proof. In all rules apart from **S**, the negative precondition merely checks whether the rule's post-condition is satisfied. Therefore, the rules are monotonic and they have a unique least fixpoint—that is, the order of rule applications is irrelevant. Furthermore, rule **S** is applied only if no other rule is applicable, and precedence ensures that the rule is applied in a deterministic way. Thus, the result of the decomposition construction algorithm is uniquely determined by $\mathcal{O}$, $\mathcal{Q}$, and $\mathcal{C}$. □

Claim 9.14. *Decomposition $\mathcal{D}$ is sound.*

Proof. The proof is by induction on rule application. After the initial step 1, we have $\mathcal{E} = \emptyset$ and $\text{core}(v) = \text{knw}(v)$ for each context v, so $\mathcal{D}$ is sound. We next consider all rules from Table 9.1.

(**H**$_k$) If $K \subseteq \text{knw}(v)$ and $K \sqsubseteq L \in \mathcal{O}$, then, by the induction hypothesis, $\mathcal{O} \models \text{core}(v) \sqsubseteq K$; hence $\mathcal{O} \models \text{core}(v) \sqsubseteq L$ follows. Consequently, adding L to $\text{knw}(v)$ preserves soundness.

(**P**$_k$) If $\langle v, u, \exists R.A\rangle \in \mathcal{E}$, $\exists R.A \in \text{knw}(v)$, and $\forall \mathsf{inv}(R).C \in \text{knw}(u)$, then, by the induction hypothesis, all of the following entailments hold:

$$\mathcal{O} \models \text{core}(v) \sqcap \exists R.A \sqsubseteq \exists R.[\text{core}(u) \sqcap A] \qquad \mathcal{O} \models \text{core}(v) \sqsubseteq \exists R.A \qquad \mathcal{O} \models \text{core}(u) \sqsubseteq \forall \mathsf{inv}(R).C$$

These entailments imply $\mathcal{O} \models \text{core}(v) \sqsubseteq \exists R.\forall \mathsf{inv}(R).C$, which then implies $\mathcal{O} \models \text{core}(v) \sqsubseteq C$. Consequently, adding C to $\text{knw}(v)$ preserves soundness.

(**S**) By the induction hypothesis, we have $\mathcal{O} \models \text{core}(v) \sqsubseteq \forall R.B$ for each $B \in \mathbf{B}_k$, which implies $\mathcal{O} \models \text{core}(v) \sqcap \exists R.A \sqsubseteq \exists R.[A \sqcap \mathbf{B}_k]$; then $\mathcal{O} \models \text{core}(v) \sqcap \exists R.A \sqsubseteq \exists R.[\text{core}(u) \sqcap A]$ follows since we have $\text{core}(u) = \text{core}' \subseteq \{A\} \cup \mathbf{B}_k$ by the requirement on strategy in Definition 9.9. Consequently, adding the edge $\langle v, u, \exists R.A\rangle$ to $\mathcal{E}$ preserves soundness.

Since no other rule affects $\mathcal{E}$ or knw, this concludes the proof of this claim. □

Claim 9.15. *Decomposition $\mathcal{D}$ is admissible.*

Proof. We check that $\mathcal{D}$ satisfies all conditions of Definition 9.3. The ontology condition and the structural conditions (S2) and (S3) hold trivially since the construction is closed under rules $\mathbf{H}_p$, $\mathbf{P}_p$, and $\mathbf{S}$, respectively.

For condition (S1), we show that $\mathrm{core}(v) \subseteq \mathrm{knw}(v)$ and $\mathrm{knw}(v) \subseteq \mathrm{poss}(v)$ hold for each context $v \in \mathcal{V}$. The first inclusion holds because each set $\mathrm{knw}(v)$ is initialized in step 1 and in rule $\mathbf{S}$ with $\mathrm{knw}(v) := \mathrm{core}(v)$, and $\mathrm{core}(v)$ remains constant during the construction while $\mathrm{knw}(v)$ only increases. The second inclusion holds because each set $\mathrm{poss}(v)$ is initialized with $\mathrm{poss}(v) := \mathrm{knw}(v)$, and the rules $\mathbf{H}_p$ and $\mathbf{P}_p$ extending $\mathrm{poss}(v)$ are applicable whenever the rules $\mathbf{H}_k$ and $\mathbf{P}_k$ extending $\mathrm{knw}(v)$ are applicable.

The initial context structure produced by initialization contains no edges, and whenever an edge $\langle v, u, \exists R.A \rangle$ is added to $\mathcal{E}$ in rule $\mathbf{S}$, strategy ensures that $\prec_u$ is R-admissible. Thus, the ordering condition holds throughout the construction.

Finally, for the covering condition, consider an arbitrary query $K \sqsubseteq M \in \mathcal{Q}$. After step 1, context $v = \vartheta(K \sqsubseteq M)$ covers $K \sqsubseteq M$ in the sense of Definition 8.6—that is, $\mathrm{core}(v) \subseteq K$ and M is $\prec_v$-minimal. Then, step 2 ensures also that $K \subseteq \mathrm{poss}(v)$; thus, v covers $K \sqsubseteq M$, as required by Definition 9.3. □

We are now ready to combine all of the results presented thus far and formulate our first result on fixed-parameter tractability of subsumption reasoning. We will further generalize this result in Theorem 10.12 in the next chapter.

Theorem 9.16. *For each control $\mathcal{C}$, the following problem is fixed-parameter tractable:*

- Inputs: *a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$ and a set of queries $\mathcal{Q}$*
- Parameter: *an integer* mwd
- Problem: *return "yes" if the $\mathcal{C}$-decomposition of $\mathcal{O}$ and $\mathcal{Q}$ has width at most* mwd*, and if also $\mathcal{O} \models K \sqsubseteq M$ holds for each query $K \sqsubseteq M \in \mathcal{Q}$*

Proof. Immediate by Theorem 9.11, and Propositions 9.7 and 9.8. □

9.5 Decompositions and the Hypertableau Algorithm

In this section we show that ϵ-free decompositions explain to an extent the performance of the hypertableau algorithm (cf. Appendix C) and are thus not specific to consequence-based algorithms. Since the hypertableau algorithm cannot answer several queries at once, we consider only the case when set $\mathcal{Q}$ contains a single query of the form $A \sqsubseteq \bot$ for A an atomic concept—that is, when our goal is to determine the satisfiability of an atomic concept. The following theorem provides us with a key insight.

Theorem 9.17. *Let $\mathcal{O}$ be a normalized $\mathcal{ALCI}$ ontology, let $q = A \sqsubseteq \bot$, let D be a derivation of the hypertableau algorithm for $\mathcal{O}$ and the ABox $\{A(a)\}$, and let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \mathrm{knw}, \mathrm{poss}, \prec, \vartheta \rangle$ be an admissible and ϵ-free decomposition of $\mathcal{O}$ and $\{q\}$. Then, for each ABox $\mathcal{A}_0$ labeling a vertex of D, there exists a mapping $\mu : \mathsf{ind}(\mathcal{A}_0) \to \mathcal{V}$ such that*

(a) if $L(s) \in \mathcal{A}_0$, then $L \in \mathrm{poss}(\mu(s))$, and

(b) if $R(s,t) \in \mathcal{A}_0$, then there exists an atomic concept A such that $\langle \mu(s), \mu(t), \exists R.A \rangle \in \mathcal{E}$.

Proof. We prove the claim by induction of the depth of D, but we strengthen property (b) to the following property (b'):

(b') if $R(s,t) \in \mathcal{A}_0$, then there exists an atomic concept A such that $\langle \mu(s), \mu(t), \exists R.A \rangle \in \mathcal{E}$ and

$$\mathrm{poss}(\mu(t)) \supseteq \{A\} \cup \{B \mid \forall R.B \in \mathrm{poss}(\mu(s))\}.$$

For the induction base, let $\mathcal{A}_0 = \{A(a)\}$ be the ABox labeling the root of D, and let $\mu(a) = \vartheta(q)$. Then $\mathcal{L}_{\mathcal{A}_0}(a) = \{A\} \subseteq \mathrm{poss}(\mu(a))$ holds by the covering condition from Definition 9.3, as required for property (a); moreover, property (b') holds vacuously. For the inductive step, let $\mathcal{A}_0$ be an ABox satisfying the property for some μ, and consider an ABox obtained from $\mathcal{A}_0$ as follows.

Assume that the Hyp-rule derives $L_j(s) \in \mathcal{A}_j$ from premises $A_1 \sqcap \ldots \sqcap A_m \sqsubseteq L_1 \sqcup \ldots \sqcup L_n \in \mathcal{O}$ and $\{A_1(s), \ldots, A_m(s)\} \subseteq \mathcal{A}_0$. Then $\{A_1, \ldots, A_m\} \subseteq \mathcal{L}_{\mathcal{A}_0}(s) \subseteq \mathrm{poss}(\mu(s))$ holds by the induction assumption; since $\mathcal{D}$ satisfies the ontology condition from Definition 9.3, $L_j \in \mathrm{poss}(\mu(s))$ follows, as required for property (a).

Assume that the $\exists$-rule derives $\{R(s,t),\ A(t)\} \subseteq A_1$ from $\exists R.A(s) \in \mathcal{A}_0$. Then $\exists R.A \in \mathrm{poss}(\mu(s))$ holds by the induction assumption, so a context $u \in \mathcal{V}$ exists that satisfies condition (S3) from Definition 9.3. We extend μ by defining $\mu(t) := u$. ABox $\mathcal{A}_1$ and the extended mapping μ then clearly satisfy properties (a) and (b').

Assume that the $\forall^+$-rule derives $B(t) \in \mathcal{A}_1$ from $\{\forall R.B(s),\ R(s,t)\} \subseteq \mathcal{A}_0$. By the induction assumption, property (a) then implies $\forall R.B \in \mathrm{poss}(\mu(s))$, and property (b') is satisfied for some atomic concept A, which immediately implies $B \in \mathrm{poss}(\mu(t))$, as required for property (a).

Assume that the $\forall^-$-rule derives $B(s) \in \mathcal{A}_1$ from $\{[\forall \mathsf{inv}(R).B](t),\ R(s,t)\} \subseteq \mathcal{A}_0$. By the induction assumption, property (a) then implies $\forall \mathsf{inv}(R).B \in \mathrm{poss}(\mu(t))$, and property (b') is satisfied for some atomic concept A. But then, condition (S2) from Definition 9.3 implies $B \in \mathrm{poss}(\mu(s))$, as required for property (a). □

Intuitively, Theorem 9.17 says that each ABox $\mathcal{A}_0$ labeling a derivation vertex can be "embedded" via some mapping μ into each admissible and ϵ-free decomposition $\mathcal{D}$ of $\mathcal{O}$ and $\mathcal{Q}$. Thus, for each individual $s \in \mathsf{ind}(\mathcal{A}_0)$, set $\mathrm{poss}(\mu(s))$ provides us with an upper bound on $\mathcal{L}_{\mathcal{A}_0}(s)$—a set used in the definition of blocking. Now let $\ell = \max_{v \in V} |\mathrm{poss}(v)|$; then, for each context in $\mathcal{D}$ we can have at most 2^ℓ different labels, so the total number of different labels is bounded by $\wp = \ln(\mathcal{D}) \cdot 2^\ell$; in other words, $\wp$ provides us with an upper bound on the number of nonblocked individuals in $\mathcal{A}_0$, which is analogous to Proposition 9.8. However, the number of indirectly blocked individuals can be exponentially larger due to dynamic blocking, but our decompositions do not analyze the computations caused by this effect.

Furthermore, assume that we modify the hypertableau algorithm in two ways: (i) we explicitly maintain the mapping μ from Theorem 9.17, and (ii) whenever the $\exists$-rule introduces a new individual t, we immediately derive $L(t)$ for each literal $L \in \mathrm{knw}(\mu(t))$. Provided that $\mathcal{D}$ is sound, these changes do not affect the soundness of the algorithm, but they ensure that also $L(s) \in \mathcal{A}_0$ holds in Theorem 9.17 for each individual $s \in \mathsf{ind}(\mathcal{A}_0)$ and each literal $L \in \mathrm{knw}(\mu(s))$. Then, the number of different labels for each context is bounded by $2^{\mathrm{wd}(\mathcal{D})}$, so the total number of labels and the number of nonblocked individuals is bounded by $\ln(\mathcal{D}) \cdot 2^{\mathrm{wd}(\mathcal{D})}$. Please note that the standard hypertableau algorithm would eventually derive all literals introduced in (ii), but since the order in which derivation rules are applied is don't-care nondeterministic, this may occur only after a substantial amount

of work. In contrast, modification (ii) eagerly introduces all known facts, which, through "earlier" blocking, can improve the algorithm's performance.

One can analogously use decompositions to obtain an automata-based algorithm running in time polynomial in $\ln(\mathcal{D})$ and $2^{\mathrm{wd}(\mathcal{D})}$, but we do not discuss the technical details any further for the sake of brevity.

There are a million ways to ask the same question,
and a million ways to answer it.

—Brian Herbert and Kevin J. Anderson
Dune: The Machine Crusade

Chapter 10

Analyzing Or-Branching Using General Decompositions

We now combine our framework with methods based on tree decompositions to also allow for a quantitative analysis of or-branching. Roughly speaking, we introduce the notion of an ϵ-refinement of an ϵ-free decomposition $\mathcal{D}$, which is a obtained from $\mathcal{D}$ by replacing each context v in $\mathcal{D}$ with a tree decomposition of the propositional problem associated with v.

In Section 10.1 we first recapitulate the notions of tree decompositions. We discuss the intuitions behind our ideas in Section 10.2, and we formalize them in Section 10.3. Finally, in Section 10.4 we show how to construct an ϵ-refinement of an ϵ-free decomposition.

10.1 Tree Decompositions and Treewidth

Tree decompositions and treewidth [105] were extensively used to obtain FPT results for a number of intractable graph-theoretic problems. For convenience, we extend these notions to hypergraphs by treating each hyperedge as the clique of all of its vertices, but this does not alter the definitions in any significant way.

Given a finite set N of nodes, an *N-hypergraph* $\mathcal{H}$ is a subset of 2^N; the elements of $\mathcal{H}$ are called *hyperedges*. A *tree decomposition* of $\mathcal{H}$ is a tuple $\mathcal{T} = \langle \mathcal{V}, \mathcal{E}, \mathcal{L} \rangle$ where $\mathcal{V}$ and $\mathcal{E}$ are sets of *vertices* and *edges*, respectively, of an undirected tree, and $\mathcal{L} : \mathcal{V} \to 2^N$ is a labeling of vertices with subsets of N such that the following conditions hold:

(T1) for each hyperedge $h \in \mathcal{H}$, a vertex $v \in \mathcal{V}$ exists such that $h \subseteq \mathcal{L}(v)$; and

(T2) for each element $n \in N$, the set $\{v \in \mathcal{V} \mid n \in \mathcal{L}(v)\}$ is connected in $\mathcal{E}$.

The *width* of $\mathcal{T}$ is $\mathrm{wd}(\mathcal{T}) := \max_{v \in \mathcal{V}} |\mathcal{L}(v)|$, and the *treewidth* of $\mathcal{H}$ is the minimum width over all tree decompositions of $\mathcal{H}$. In related literature width is actually defined as $\mathrm{wd}(\mathcal{T}) := \max_{v \in \mathcal{V}} |\mathcal{L}(v)| - 1$ in order to ensure that tree-like hypergraphs have width one, but we drop the term -1 from our definition for uniformity with Chapter 9. For a fixed integer mwd, a tree decomposition of width at most mwd can be computed in linear time [22]. This result was originally formulated for tree decompositions of graphs; however, by treating hyperedges as cliques, this result can also be straightforwardly applied to our definitions.

Lemma 10.1 ([22])**.** *Let $\mathcal{H}$ be an N-hypergraph, and let* mwd *be an integer. One can compute a tree decomposition of $\mathcal{H}$ of width at most* mwd*, or determine that such a tree decomposition does not exist, in time $O(f(\mathrm{mwd}) \cdot |N|)$ where f is a computable function. The resulting tree decomposition has at most $|\mathcal{H}|$ vertices.*

In the rest of this section we recapitulate an FPT procedure for propositional satisfiability [126] based on these notions. To unify the notation throughout this thesis, we write propositional clauses as implications $K \sqsubseteq M$ where K is a conjunction of atoms, and M is a disjunction of atoms; furthermore, we consider a propositional interpretation J to be a set of atoms; finally, we write $J \models K \sqsubseteq M$ if $K \subseteq J$ implies $M \cap J \neq \emptyset$.

Let $\mathcal{N}$ be a set of propositional clauses. We define $\mathcal{H}$ as the hypergraph whose nodes are the atoms occurring in $\mathcal{N}$ and that contains a hyperedge $K \cup M \in \mathcal{H}$ for each clause $K \sqsubseteq M \in \mathcal{N}$. A tree decomposition and the treewidth of $\mathcal{N}$ are defined as a tree decomposition and the treewidth, respectively, of $\mathcal{H}$. Now assume that the treewidth of $\mathcal{N}$ is bounded by some integer mwd. The following procedure checks the satisfiability of $\mathcal{N}$ and is fixed-parameter tractable w.r.t. mwd.

1. Compute a tree decomposition $\mathcal{T} = \langle \mathcal{V}, \mathcal{E}, \mathcal{L} \rangle$ of $\mathcal{H}$ of width at most mwd. By Lemma 10.1, this step is FPT w.r.t. mwd.

2. Associate with each vertex $v \in \mathcal{V}$ a set $\mathcal{P}_v$ of propositional interpretations as follows: initially set $\mathcal{P}_v := 2^{\mathcal{L}(v)}$, and then eliminate each $J \in \mathcal{P}_v$ for which a clause $K \sqsubseteq M \in \mathcal{N}$ exists such that

$K \cup M \subseteq \mathcal{L}(v)$ and $J \not\models K \sqsubseteq M$. Since $|\mathcal{L}(v)| \leq \text{mwd}$, this step requires time polynomial in 2^{mwd} and $|\mathcal{N}|$, and is thus FPT w.r.t. mwd.

3. Traversing through $\mathcal{T}$ in a bottom-up way starting from the leaves towards the root, for each non-root vertex v and its parent v', eliminate from $\mathcal{P}_{v'}$ each $J' \in \mathcal{P}_{v'}$ for which no $J \in \mathcal{P}_v$ exists such that $J \cap \mathcal{L}(v) \cap \mathcal{L}(v') = J' \cap \mathcal{L}(v) \cap \mathcal{L}(v')$. We can achieve this by iterating over all pairs of $J' \in \mathcal{P}_{v'}$ and $J \in \mathcal{P}_v$; since $\mathcal{T}$ has at most $|\mathcal{N}|$ vertices by Lemma 10.1, this step is FPT w.r.t. mwd.

4. Set $\mathcal{N}$ is unsatisfiable if and only if $\mathcal{P}_r = \emptyset$, where r is the root of $\mathcal{T}$.

To see why this algorithm is correct, assume that $\mathcal{P}_r \neq \emptyset$; we can then construct an interpretation J for $\mathcal{N}$ as follows. Select an arbitrary propositional interpretation $J_r \in \mathcal{P}_r$; since J_r has not been deleted in step 3, for $v_1, \ldots, v_k$ the children of r, we can select interpretations $J_{v_1}, \ldots, J_{v_k}$ such that $J_r \cap \mathcal{L}(r) \cap \mathcal{L}(v_i) = J_{v_i} \cap \mathcal{L}(r) \cap \mathcal{L}(v_i)$ holds for each $1 \leq i \leq k$. By inductively repeating this argument from the root to the leaves, we associate with each vertex $v \in \mathcal{V}$ an interpretation J_v, and then we define $J = \bigcup_{v \in \mathcal{V}} J_v$. Now since $\mathcal{T}$ satisfies properties (T1) and (T2), it is straightforward to see that $J \models \mathcal{N}$. The converse direction can be shown analogously, and we omit it for the sake of brevity. Thus, the treewidth of $\mathcal{N}$ can be taken as an indicator of the "hardness" of $\mathcal{N}$, and it shows the size of the sets of atomic concepts that we need to consider to decide the satisfiability of $\mathcal{N}$.

10.2 Intuitions

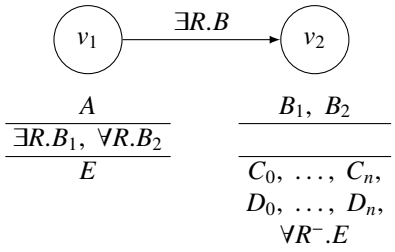
Example 10.2. Let $\mathcal{O}$ be the ontology consisting of axioms (10.1)–(10.7), and let $q = A \sqsubseteq E$. One can readily check that $\mathcal{O} \models q$ holds.

$$A \sqsubseteq \exists R.B_1 \quad (10.1) \qquad B_1 \sqcap B_2 \sqsubseteq C_0 \sqcup D_0 \quad (10.3) \qquad C_n \sqsubseteq \forall R^-.E \quad (10.6)$$

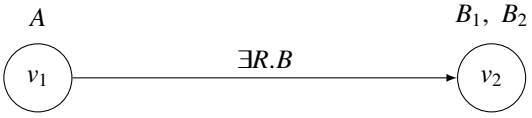
$$A \sqsubseteq \forall R.B_2 \quad (10.2) \qquad C_{i-1} \sqsubseteq C_i \sqcup D_i \quad \text{for } 1 \leq i \leq n \quad (10.4) \qquad D_n \sqsubseteq \forall R^-.E \quad (10.7)$$

$$D_{i-1} \sqsubseteq C_i \sqcup D_i \quad \text{for } 1 \leq i \leq n \quad (10.5)$$

The $\mathcal{C}$-decomposition $\mathcal{D}_1$ of $\mathcal{O}$ and q obtained using the eager strategy is shown in Figure 10.1a, and the inferences of Algorithm 9.4 on $\mathcal{D}_1$ are shown in Figure 10.1b. Clause (14) stands for clauses with $M \subseteq \{C_0, \ldots, C_n, D_0, \ldots, D_n, \forall R^-.E\}$; exactly which clauses of this form are derived depends



(a) Decomposition $\mathcal{D}_1$



Initialization:	$\top \sqsubseteq A$	(8)	Initialization:	$\top \sqsubseteq B_1$	(9)
H[10.1+8]:	$\top \sqsubseteq \exists R.B_1$	(11)	Initialization:	$\top \sqsubseteq B_2$	(10)
H[10.2+8]:	$\top \sqsubseteq \forall R.B_2$	(12)	H[10.3+9+10]:	$\top \sqsubseteq C_0 \sqcup D_0$	(13)
P[15]:	$\top \sqsubseteq E$	(16)		...	
			H:	$\top \sqsubseteq M$	(14)
				...	
			H:	$\top \sqsubseteq \forall R^-.E$	(15)

(b) Inferences

Figure 10.1: An ϵ-free decomposition of $\mathcal{O}$ and q and the related inferences

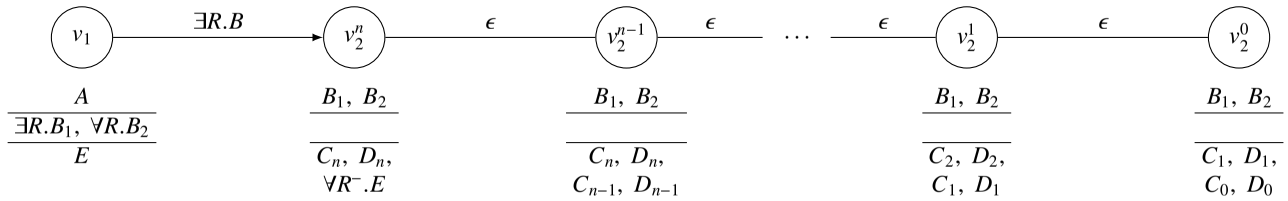
on the literal ordering $\prec_{v_2}$. For example, with $\forall R^-.E \prec_{v_2} C_n \prec_{v_2} D_n \prec_{v_2} \ldots \prec_{v_2} C_0 \prec_{v_2} D_0$ we derive a linear number of clauses, and with $\forall R^-.E \prec_{v_2} C_0 \prec_{v_2} D_0 \prec_{v_2} \ldots \prec_{v_2} C_n \prec_{v_2} D_n$ we derive an exponential number of clauses. In practice, it is difficult to identify a literal ordering that minimizes the number of derived clauses, so the only guarantee on the algorithm's performance is exponential in n; this is in agreement with the fact that $\mathrm{wd}(\mathcal{D}_1) = 2n + 3$.

Clauses in $\mathcal{S}(v_2)$ essentially solve a propositional problem consisting of clauses (10.3)–(10.7) and clause $\alpha = B_1 \sqcap B_2 \sqsubseteq \forall R^-.E$: the antecedent of α captures the information that can be propagated from v_1 to v_2, and the consequent of α captures the information that can be propagated from v_2 to v_1. As we explained in Section 10.1, the treewidth of propositional problems serves as a measure of the difficulty of such problems, and the bottom-up algorithm provides an FPT decision procedure for propositional satisfiability. Thus, to explain the difficulty of or-branching in description logics, we next incorporate tree decompositions into our framework. To this end, we replace context v_2 with a tree decomposition of the mentioned propositional problem, where we label the newly introduced edges by a special symbol ϵ; similarly to edges in a tree decomposition, our ϵ-edges will always be symmetric. Decomposition $\mathcal{D}_2$ obtained in this way is shown in Figure 10.2a. The set of

contexts $\{v_2^0, \ldots, v_2^n\}$ obtained by replacing v_2 is called an *ϵ-component* of $\mathcal{D}_2$. Intuitively, each ϵ-component $\mathcal{W}$ of $\mathcal{D}_2$ corresponds to one or more domain elements in a model of $\mathcal{O}$, and each context $v \in \mathcal{W}$ provides a partial interpretation for the literals in poss(v). Condition (N2) of Definition 9.2 requires all contexts in an ϵ-component $\mathcal{W}$ of a sound decomposition to have the same core; thus, all partial descriptions of the elements represented by $\mathcal{W}$ must coincide on the "assumed" literals. However, the contexts in $\mathcal{W}$ can differ on known and possible literals, which will allow us to reduce decomposition width.

We will extend the notion of decomposition admissibility so that it guarantees completeness of a modified consequence-based algorithm shown in Table 10.1. The Hyper and Pred rules are as in Table 8.1, but modified so that a clause is added to some $\mathcal{S}(v)$ only if all of its literals are possible at context v. In addition, we introduce a new Epsilon rule, which essentially performs hyperresolution across ϵ-edges: given an edge $\langle v, u, \epsilon \rangle$, a clause in $\bigsqcap_{i=1}^n L_i \sqsubseteq M \in \mathcal{S}(u)$ is chosen such that all literals in M are possible at context v, each literal L_i is resolved away using a clause in $\mathcal{S}(v)$, and the result is added to $\mathcal{S}(v)$. The Epsilon rule can be seen as a practical variant of the bottom-up algorithm for propositional satisfiability described in Section 10.1.

The inferences of our new algorithm on $\mathcal{O}$, q, and $\mathcal{D}_2$ are shown in Figure 10.2b. As before, we first initialize each $\mathcal{S}(v)$ with $\top \sqsubseteq L$ for each literal L known at context v, thus obtaining clauses (17)–(27). Moreover, for each edge $\langle v, u, \epsilon \rangle$ and each literal L that is possible at both u and v, we initialize $\mathcal{S}(u)$ and $\mathcal{S}(v)$ with $L \sqsubseteq L$, thus obtaining clauses (28)–(39). Finally, using rules from Table 10.1, we derive clauses (40)–(52). At each context v_2^i with $0 \leq i < n$, inferences follow the same pattern: using clauses (10.4) and (10.5) from $\mathcal{O}$, we derive a clause of the form $\top \sqsubseteq C_{i+1} \sqcup D_{i+1}$, which the Epsilon rule then "transfers" over an ϵ-edge into context v_2^{i+1}. Finally, in context v_2^n we derive clause (51), and then by the Pred rule we derive the goal clause (52). In order to ensure that the Epsilon rule derives all relevant consequences, our notion of decomposition admissibility imposes an additional restriction on the literal orderings of contexts. For example, clause (42) is relevant for the Epsilon rule and is derived using the Hyper rule from clause (41); however, to facilitate the latter derivation, concepts D_1 and C_1 must not be larger in $\prec_{v_2^0}$ than concept D_0. Thus, the new restriction on literal orderings is analogous to R-admissibility for $\exists R.A$-edges, but adapted to ϵ-edges. To estimate the complexity of our algorithm, we note that $\mathrm{wd}(\mathcal{D}_2) = 4$; hence, the number of clauses derived at each context is bounded by $4^{\mathrm{wd}(\mathcal{D}_2)} = 4^4$, and this bound does not depend on



(a) Decomposition $\mathcal{D}_2$

I:	$\top \sqsubseteq A$	(25)
I:	$\top \sqsubseteq \exists R.B_1$	(26)
I:	$\top \sqsubseteq \forall R.B_2$	(27)
P[51]:	$\top \sqsubseteq E$	(52)

I:	$\top \sqsubseteq B_1$	(23)
I:	$\top \sqsubseteq B_2$	(24)
I:	$C_n \sqsubseteq C_n$	(38)
I:	$D_n \sqsubseteq D_n$	(39)
E[48]:	$\top \sqsubseteq C_n \sqcup D_n$	(49)
H[10.6+49]:	$\top \sqsubseteq \forall R^-.E \sqcup D_n$	(50)
H[10.7+50]:	$\top \sqsubseteq \forall R^-.E$	(51)

I:	$\top \sqsubseteq B_1$	(21)
I:	$\top \sqsubseteq B_2$	(22)
I:	$C_{n-1} \sqsubseteq C_{n-1}$	(34)
I:	$D_{n-1} \sqsubseteq D_{n-1}$	(35)
I:	$C_n \sqsubseteq C_n$	(36)
I:	$D_n \sqsubseteq D_n$	(37)
E[...]:	$\top \sqsubseteq C_{n-1} \sqcup D_{n-1}$	(46)
H[10.4+46]:	$\top \sqsubseteq C_n \sqcup D_n \sqcup D_{n-1}$	(47)
H[10.5+47]:	$\top \sqsubseteq C_n \sqcup D_n$	(48)

I:	$\top \sqsubseteq B_1$	(19)
I:	$\top \sqsubseteq B_2$	(20)
I:	$C_1 \sqsubseteq C_1$	(30)
I:	$D_1 \sqsubseteq D_1$	(31)
I:	$C_2 \sqsubseteq C_2$	(32)
I:	$D_2 \sqsubseteq D_2$	(33)
E[42]:	$\top \sqsubseteq C_1 \sqcup D_1$	(43)
H[10.4+43]:	$\top \sqsubseteq C_2 \sqcup D_2 \sqcup D_1$	(44)
H[10.5+44]:	$\top \sqsubseteq C_2 \sqcup D_2$	(45)

I:	$\top \sqsubseteq B_1$	(17)
I:	$\top \sqsubseteq B_2$	(18)
I:	$C_1 \sqsubseteq C_1$	(28)
I:	$D_1 \sqsubseteq D_1$	(29)
H[10.3+17+18]:	$\top \sqsubseteq C_0 \sqcup D_0$	(40)
H[10.4+40]:	$\top \sqsubseteq C_1 \sqcup D_1 \sqcup D_0$	(41)
H[10.5+41]:	$\top \sqsubseteq C_1 \sqcup D_1$	(42)

(b) Inferences

Figure 10.2: A decomposition of $\mathcal{O}$ and q with ϵ-edges and the related inferences

the literal ordering or the number n. In this way, $\mathrm{wd}(\mathcal{D}_2)$ provides us with a better estimate of the or-branching involved in deciding $\mathcal{O} \models q$.

As in Chapter 9, the definitions of soundness and admissibility for general decompositions do not directly provide us with an algorithm for computing decompositions. As a possible remedy, in Section 10.4 we introduce the notion of an *ϵ-refinement*, which is obtained from an ϵ-free decomposition by replacing each context v with a tree decomposition of the propositional problem associated with v. In our example, $\mathcal{D}_2$ is an ϵ-refinement of $\mathcal{D}_1$. We will show that computing an ϵ-refinement involves determining a linear number of tree decompositions and is thus fixed-parameter tractable w.r.t. decomposition width.

10.3 Formalization

Decompositions and decomposition soundness (with or without ϵ-edges) were already introduced in Section 9.3, so we next focus on admissibility of decompositions with ϵ-edges. In the rest of this section we fix a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$ and a finite set of queries $\mathcal{Q}$; furthermore, we let $\mathbf{L}$ be the set of literals that occur in $\mathcal{O} \cup \mathcal{Q}$.

Definition 10.3. Let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \mathrm{knw}, \mathrm{poss}, \prec, \vartheta \rangle$ be a decomposition of $\mathcal{O}$ and $\mathcal{Q}$, and let $\mathcal{W}$ be an arbitrary set such that $\mathcal{W} \subseteq \mathcal{V}$. Then, let $\mathrm{poss}(\mathcal{W}) \coloneqq \bigcup_{w \in \mathcal{W}} \mathrm{poss}(w)$, and let $\mathrm{knw}(\mathcal{W})$ and $\mathrm{core}(\mathcal{W})$ be defined analogously. The *ϵ-projection* of $\mathcal{D}$ w.r.t. $\mathcal{W}$, written $\mathcal{D}_{\mathcal{W}}$, is the graph whose set of vertices is $\mathcal{W}$ and that contains the undirected edge $\{u, v\}$ for each $\langle u, v, \epsilon \rangle \in \mathcal{E}$ with $u, v \in \mathcal{W}$. Set $\mathcal{W}$ is *ϵ-connected* if $\mathcal{D}_{\mathcal{W}}$ is connected; furthermore, $\mathcal{W}$ is an *ϵ-component* of $\mathcal{D}$ if $\mathcal{W}$ is ϵ-connected, and no ϵ-connected set of vertices $\mathcal{W}'$ exists such that $\mathcal{W} \subsetneq \mathcal{W}' \subseteq \mathcal{V}$.

Let $v \in \mathcal{V}$ be an arbitrary context of $\mathcal{D}$, let $\mathcal{W}$ be the ϵ-component of $\mathcal{D}$ such that $v \in \mathcal{W}$, and let $K \sqsubseteq M$ be an arbitrary query; then, v is *sound* for $K \sqsubseteq M$ if $\mathrm{core}(v) \subseteq K$; v is *complete* for $K \sqsubseteq M$ if $K \subseteq \mathrm{poss}(v)$, $M \cap \mathrm{poss}(\mathcal{W}) \subseteq \mathrm{poss}(v)$, and M is $\prec_v$-minimal; and v *covers* $K \sqsubseteq M$ if v is both sound and complete for $K \sqsubseteq M$. Finally, $\mathcal{D}$ is *admissible* if all of the following conditions are satisfied.

- Epsilon conditions:

  (E1) For each $\langle w, u, \epsilon \rangle \in \mathcal{E}$, we have $\langle u, w, \epsilon \rangle \in \mathcal{E}$.

  (E2) For each ϵ-component $\mathcal{W}$ of $\mathcal{D}$, graph $\mathcal{D}_{\mathcal{W}}$ is an undirected tree.

(E3) For each ϵ-component $\mathcal{W}$ of $\mathcal{D}$ and each literal $L \in \text{poss}(\mathcal{W})$, set $\{w \in \mathcal{W} \mid L \in \text{poss}(w)\}$ is ϵ-connected.

- Structural conditions:

(S1) For each context $v \in \mathcal{V}$, we have $\text{core}(v) \subseteq \text{knw}(v) \subseteq \text{poss}(v)$.

(S2) For each ϵ-component $\mathcal{U}$ of $\mathcal{D}$, each edge $\langle w, u, \exists R.A\rangle \in \mathcal{E}$ with $u \in \mathcal{U}$, and each literal $\forall \mathsf{inv}(R).C \in \text{poss}(\mathcal{U})$, we have $C \in \text{poss}(w)$ and $\forall \mathsf{inv}(R).C \in \text{poss}(u)$.

(S3) For each ϵ-component $\mathcal{W}$ of $\mathcal{D}$ and each literal $\exists R.A \in \text{poss}(\mathcal{W})$, there exists an edge $\langle w, u, \exists R.A\rangle \in \mathcal{E}$ with $w \in \mathcal{W}$ such that

- $\text{poss}(w) \supseteq \{\exists R.A\} \cup \{\forall R.B \mid \forall R.B \in \text{poss}(\mathcal{W})\}$, and
- $\text{poss}(u) \supseteq \{A\} \cup \{B \mid \forall R.B \in \text{poss}(\mathcal{W})\}$.

- Ordering conditions:

(P1) For each $\exists R.A$-edge $\langle w, u, \exists R.A\rangle \in \mathcal{E}$, literal ordering $\prec_u$ is R-admissible.

(P2) For each ϵ-edge $\langle w, u, \epsilon\rangle \in \mathcal{E}$, set $\text{poss}(w) \cap \text{poss}(u)$ is $\prec_u$-minimal.

- Ontology condition: For each ϵ-component $\mathcal{W}$ of $\mathcal{D}$ and each clause $K \sqsubseteq M \in \mathcal{O}$ satisfying $K \subseteq \text{poss}(\mathcal{W})$, there exists a context $w \in \mathcal{W}$ such that $K \cup M \subseteq \text{poss}(w)$.

- Covering condition: Each query $q \in \mathcal{Q}$ is covered in the context $\vartheta(q)$.

Definition 10.3 can be intuitively understood as follows. The set of contexts in $\mathcal{D}$ can be partitioned into ϵ-components—maximal sets of contexts connected by ϵ-edges. Each ϵ-component can be understood as a tree decomposition of a propositional problem, so conditions (E1)–(E3) require the contexts in the ϵ-component to form an undirected tree satisfying the connectedness condition (T2) of tree decompositions. Structural conditions (S1)–(S3) extend the respective conditions from Definition 9.3. To understand condition (S2), let $\langle w, u, \exists R.A\rangle$ be an arbitrary edge of $\mathcal{D}$, let $\mathcal{U}$ be the ϵ-component of $\mathcal{D}$ that u belongs to, and assume that $\forall \mathsf{inv}(R).C$ is possible at some context $u' \in \mathcal{U}$ different from u. Since u' and u describe the same elements in a model of $\mathcal{O}$, the elements represented by context v might need to satisfy C, which is taken care of by the Pred rule. To ensure that the rule can be applied to edge $\langle w, u, \exists R.A\rangle$, condition (S2) requires $\forall \mathsf{inv}(R).C$ to be possible

Hyper	If	$\bigsqcap_{i=1}^{n} A_i \sqsubseteq M \in \mathcal{O}$ with $\underline{M \subseteq \mathsf{poss}(v)}$,
		$K_i \sqsubseteq M_i \sqcup A_i \in \mathcal{S}(v)$ with $A_i \not\prec_v M_i$ for $1 \leq i \leq n$,
		and $\bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i \notin \mathcal{S}(v)$,
	then	add $\bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i$ to $\mathcal{S}(v)$.
Pred	If	$\langle v, u, \exists R.A \rangle \in \mathcal{E}$,
		$A \sqcap \bigsqcap_{i=1}^{n} B_i \sqsubseteq \bigsqcup_{j=1}^{m} \forall \mathsf{inv}(R).C_j \in \mathcal{S}(u)$ or $\bigsqcap_{i=1}^{n} B_i \sqsubseteq \bigsqcup_{j=1}^{m} \forall \mathsf{inv}(R).C_j \in \mathcal{S}(u)$
		with $\underline{C_j \in \mathsf{poss}(v) \text{ for each } 1 \leq j \leq m}$,
		$K_0 \sqsubseteq M_0 \sqcup \exists R.A \in \mathcal{S}(v)$ with $\exists R.A \not\prec_v M_0$,
		$K_i \sqsubseteq M_i \sqcup \forall R.B_i \in \mathcal{S}(v)$ with $\forall R.B_i \not\prec_v M_i$ for $1 \leq i \leq n$,
		and $\bigsqcap_{i=0}^{n} K_i \sqsubseteq \bigsqcup_{i=0}^{n} M_i \sqcup \bigsqcup_{j=1}^{m} C_j \notin \mathcal{S}(v)$,
	then	add $\bigsqcap_{i=0}^{n} K_i \sqsubseteq \bigsqcup_{i=0}^{n} M_i \sqcup \bigsqcup_{j=1}^{m} C_j$ to $\mathcal{S}(v)$.
Epsilon	If	$\langle v, u, \epsilon \rangle \in \mathcal{E}$,
		$\bigsqcap_{i=1}^{n} L_i \sqsubseteq M \in \mathcal{S}(u)$ with $\underline{M \subseteq \mathsf{poss}(v)}$,
		$K_i \sqsubseteq M_i \sqcup L_i \in \mathcal{S}(v)$ with $L_i \not\prec_v M_i$ for $1 \leq i \leq n$,
		and $\bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i \notin \mathcal{S}(v)$,
	then	add $\bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i$ to $\mathcal{S}(v)$.

Table 10.1: Consequence-based rules for decompositions with ϵ-edges

at u, and C to be possible at v (just like in Definition 9.3). To understand condition (S3), let $\mathcal{W}$ be an arbitrary ϵ-component of $\mathcal{D}$, assume that $\exists R.A$ is possible at some context $w' \in \mathcal{W}$, and assume that $\mathcal{D}$ contains an edge $\langle w, u, \exists R.A \rangle$ with w different from w'. Contexts w and w' jointly describe elements of a model of $\mathcal{O}$, so the existential restriction $\exists R.A$ also "applies" to w and the mentioned edge as well; thus, all literals $\forall R.B$ possible at $\mathcal{W}$ must be possible at w, and all corresponding concepts B must be possible at u in order to satisfy the semantics of $\exists R.A$-edges. The ontology condition is modified in similar vein; in particular, assume that $\mathcal{D}$ contains an ϵ-component $\mathcal{W}$ such that each antecedent literal of a clause $K \sqsubseteq M \in \mathcal{O}$ is possible at some context in $\mathcal{W}$; then, the domain elements represented by $\mathcal{W}$ may satisfy K; but then, we must find a context $w \in \mathcal{W}$ in which we can apply the Hyper rule to $K \sqsubseteq M$. The notion of covering is extended analogously. Finally, the ordering condition restricts the literal ordering as we discussed in Section 10.2. Please note that the notions of admissibility introduced in Definitions 9.3 and 10.3 coincide on ϵ-free decompositions.

To generalize our consequence-based algorithm as described in Section 10.2, we modify the Hyper and the Pred rules as shown in Table 10.1 (the modifications are underlined), and we add a new Epsilon rule. Theorem 10.4 provides us with the completeness claim for the new algorithm; the proof is technically involved, so we defer it to Appendix D.2. In addition, Proposition 10.5 provides

us with a matching soundness claim.

Theorem 10.4 (Completeness)**.** *Let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \mathrm{knw}, \mathrm{poss}, \prec, \vartheta \rangle$ be an admissible decomposition of $\mathcal{O}$ and $\mathcal{Q}$, and let $\mathcal{S}$ be a clause system for $\mathcal{D}$ such that*

(I1) no inference rule in Table 10.1 is applicable to $\mathcal{S}$,

(I2) $\top \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ for each context $v \in \mathcal{V}$ and each literal $L \in \mathrm{knw}(v)$,

(I3) $B \sqsubseteq B \mathrel{\hat{\in}} \mathcal{S}(u)$ for each edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ and each concept $B \in \mathrm{poss}(u)$ such that either $B = A$ or $\forall R.B \in \mathrm{poss}(v)$, and

(I4) $L \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(u)$ for each edge $\langle v, u, \epsilon \rangle \in \mathcal{E}$ and each literal $L \in \mathrm{poss}(v) \cap \mathrm{poss}(u)$.

Then, $K \sqsubseteq M \mathrel{\hat{\in}} \mathcal{S}(v)$ holds for each query $K \sqsubseteq M$ and each context $v \in \mathcal{V}$ that satisfy all of the following three conditions:

- *$\mathcal{O} \models K \sqsubseteq M$,*
- *context v is complete for query $K \sqsubseteq M$, and*
- *$K \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ for each literal $L \in K$.*

Proposition 10.5 (Soundness)**.** *Let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \prec \rangle$ be a sound decomposition of $\mathcal{O}$ and $\mathcal{Q}$, and let $\mathcal{S}_1$ be a clause system for $\mathcal{D}$ sound for $\mathcal{O}$. Then, each clause system $\mathcal{S}_2$ obtained by applying an inference rule from Table 10.1 to $\mathcal{D}$ and $\mathcal{S}_1$ is sound for $\mathcal{O}$.*

Proof. Rules Hyper and Pred in Table 10.1 are more restrictive than the corresponding rules in Table 8.1, so the proof of Proposition 8.8 applies. Assume that the Epsilon rule is applied as shown in Table 10.1, so we show that $\mathcal{O} \models \mathrm{core}(v) \sqcap \bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i$. To this end, let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ be an arbitrary model of $\mathcal{O}$ and consider an arbitrary element $\delta \in \Delta^{\mathcal{I}}$ such that $\delta \in [\mathrm{core}(v) \sqcap \bigsqcap_{i=1}^{n} K_i]^{\mathcal{I}}$. Now $\mathcal{S}_1$ is sound for $\mathcal{O}$, so, for each $1 \leq i \leq n$, we have $\mathcal{I} \models \mathrm{core}(v) \sqcap K_i \sqsubseteq M_i \sqcup L_i$, which together with our assumption implies $\delta \in (M_i \sqcup L_i)^{\mathcal{I}}$. If $\delta \in M_i^{\mathcal{I}}$ for some $1 \leq i \leq n$, then $\delta \in (\bigsqcup_{i=1}^{n} M_i)^{\mathcal{I}}$. Otherwise, we have $\delta \in L_i^{\mathcal{I}}$ for each $1 \leq i \leq n$; but then, $\mathcal{I} \models \mathrm{core}(u) \sqcap \bigsqcap_{i=1}^{n} L_i \sqsubseteq M$ holds by the induction assumption and the fact that $\mathcal{S}_1$ is sound for $\mathcal{O}$, and $\mathrm{core}(v) = \mathrm{core}(u)$ holds by the soundness condition (N2); together, these observations imply $\delta \in M^{\mathcal{I}}$. Either way, $\delta \in (M \sqcup \bigsqcup_{i=1}^{n} M_i)^{\mathcal{I}}$ holds. Since δ was chosen arbitrarily, we have $\mathcal{I} \models \mathrm{core}(v) \sqcap \bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i$, as required. □

We next extend the consequence-based algorithm for ϵ-free decompositions (i.e., Algorithm 9.4) so that it can be applied to decompositions with ϵ-edges. Apart from applying the modified rules from Table 10.1, the only other difference is in step 3 which is needed for the Epsilon rule. The Elim rule and redundancy elimination are the same as in Section 8.4.

Algorithm 10.6. The *decomposition algorithm* for $\mathcal{ALCI}$ takes $\mathcal{O}$, $\mathcal{Q}$, and a sound and admissible decomposition $\mathcal{D}$ of $\mathcal{O}$ and $\mathcal{Q}$. The algorithm (nondeterministically) computes a clause system $\mathcal{S}$ for $\mathcal{D}$ as follows.

1. Set $\mathcal{S}(v) := \{\top \sqsubseteq L \mid L \in \mathrm{knw}(v)\}$ for each context $v \in \mathcal{V}$.

2. For each edge $\langle v, u, \exists R.A\rangle \in \mathcal{E}$ and each concept $B \in \mathrm{uknw}(u)$ such that $B = A$ or $\forall R.B \in \mathrm{poss}(v)$, add $B \sqsubseteq B$ to $\mathcal{S}(u)$.

3. For each ϵ-edge $\langle v, u, \epsilon\rangle \in \mathcal{E}$ and each literal $L \in \mathrm{poss}(v) \cap \mathrm{uknw}(u)$, add $L \sqsubseteq L$ to $\mathcal{S}(u)$.

4. For each query $K \sqsubseteq M \in \mathcal{Q}$, let $v := \vartheta(K \sqsubseteq M)$; then, for each literal $L \in K \setminus \mathrm{knw}(v)$, add $L \sqsubseteq L$ to $\mathcal{S}(v)$.

5. Exhaustively apply rules Hyper, Pred, and Epsilon from Table 10.1 and rule Elim from Table 8.1.

Lemma 8.11 (showing that if $K \sqsubseteq M \mathrel{\hat{\in}} \mathcal{S}(v)$ holds at some point during algorithm's execution, then this also holds at all future points) clearly applies to Algorithm 10.6 as well: the Epsilon rule only adds clauses, and the Elim rule is the same as in Section 8.3. Furthermore, the proof of Lemma 9.5 (showing that each clause in $\mathcal{S}(v)$ is of the form $\top \sqsubseteq L$ with $L \in \mathrm{knw}(v)$ or of the form $K \sqsubseteq M$ with $K \cup M \subseteq \mathrm{uknw}(v)$) can be straightforwardly extended to Algorithm 10.6. This allows us to establish in Proposition 10.7 the soundness and completeness of our algorithm, and then determine in Proposition 10.8 the algorithm's complexity.

Proposition 10.7. *Let $\mathcal{S}$ be a clause system obtained by applying Algorithm 10.6 to $\mathcal{O}$, $\mathcal{Q}$, and $\mathcal{D}$. Then, for each query $q \in \mathcal{Q}$, we have $\mathcal{O} \models q$ if and only if $q \mathrel{\hat{\in}} \mathcal{S}(\vartheta(q))$.*

Proof. By Lemma 8.11, steps 2–4 ensure conditions (I2)–(I4) and step 5 ensures the required condition on the queries in Theorem 10.4; then, the claim of this proposition follows from Theorem 10.4 and Proposition 10.5 as in the proof of Proposition 9.7. □

Proposition 10.8 (Termination)**.** *Algorithm 10.6 terminates in time polynomial in* $4^{\mathrm{wd}(\mathcal{D})^2}$*,* $\ln(\mathcal{D})$*, and* $\|\mathcal{O}\| + \|\mathcal{Q}\|$*.*

Proof. Analogous to the proof of Proposition 9.8. □

10.4 Constructing Decompositions via ϵ-Refinement

We next formalize the notion of ϵ-refinement of an ϵ-free decomposition $\mathcal{D}$ that we described in Section 10.2.

Definition 10.9. Let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \mathrm{knw}, \mathrm{poss}, \prec, \vartheta \rangle$ be an ϵ-free decomposition of $\mathcal{O}$ and $\mathcal{Q}$. For each edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$, sets $\delta_{v,u}^{\exists R.A}$ and $\varrho_{v,u}^{\exists R.A}$ are defined as follows:

$$\delta_{v,u}^{\exists R.A} := \{\exists R.A\} \cup \{\forall R.B \mid \forall R.B \in \mathrm{poss}(v)\} \cup \{C \mid \forall \mathsf{inv}(R).C \in \mathrm{poss}(u)\}$$

$$\varrho_{v,u}^{\exists R.A} := \{A\} \cup \{B \mid \forall R.B \in \mathrm{poss}(v)\} \cup \{\forall \mathsf{inv}(R).C \mid \forall \mathsf{inv}(R).C \in \mathrm{poss}(u)\}$$

Function $\mathcal{H}$ for $\mathcal{D}$ maps each context $v \in \mathcal{V}$ to an $\mathbf{L}$-hypergraph $\mathcal{H}_v$ such that each $\mathcal{H}_v$ is the smallest set satisfying all of the following conditions:

(R1) $(K \cup M) \cap \mathrm{uknw}(v) \in \mathcal{H}_v$ for each $v \in \mathcal{V}$ and each $K \sqsubseteq M \in \mathcal{O}$ satisfying $K \cup M \subseteq \mathrm{poss}(v)$,

(R2) $\delta_{v,u}^{\exists R.A} \cap \mathrm{uknw}(v) \in \mathcal{H}_v$ and $\varrho_{v,u}^{\exists R.A} \cap \mathrm{uknw}(u) \in \mathcal{H}_u$ for each edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$, and

(R3) $(K \cup M) \cap \mathrm{uknw}(v) \in \mathcal{H}_v$ for each query $q = K \sqsubseteq M \in \mathcal{Q}$ and context $v = \vartheta(q)$.

A decomposition $\mathcal{D}' = \langle \mathcal{V}', \mathcal{E}', \mathrm{core}', \mathrm{knw}', \mathrm{poss}', \prec', \vartheta' \rangle$ of $\mathcal{O}$ and $\mathcal{Q}$ is an *ϵ-refinement* of $\mathcal{D}$ if there exists a function $\mathcal{T}$ mapping each context $v \in \mathcal{V}$ to a tree decomposition $\mathcal{T}_v = \langle \mathcal{V}_v, \mathcal{E}_v, \mathcal{L}_v \rangle$ of $\mathcal{H}_v$ such that $\mathcal{V}_v \cap \mathcal{V}_w = \emptyset$ and $\mathcal{V}_v \subseteq \mathbf{X}$ for all $v, w \in \mathcal{V}$ with $v \neq w$, and all of the following conditions are satisfied.

(R4) $\mathcal{V}' = \bigcup_{v \in \mathcal{V}} \mathcal{V}_v$.

(R5) For each context $v \in \mathcal{V}$ and each vertex $w \in \mathcal{V}_v$, we have

$$\mathrm{core}'(w) = \mathrm{core}(v), \qquad \mathrm{knw}'(w) = \mathrm{knw}(v), \qquad \textit{and} \qquad \mathrm{poss}'(w) = \mathrm{knw}(v) \cup \mathcal{L}_v(w).$$

(R6) For all $v, u \in \mathcal{V}$, each $w \in \mathcal{V}_v$, and each $z \in \mathcal{V}_u$, we have $\langle w, z, \epsilon \rangle \in \mathcal{E}'$ if and only if $v = u$ and $\{w, z\} \in \mathcal{E}_v$.

(R7) For all $v, u \in \mathcal{V}$, each $w \in \mathcal{V}_v$, each $z \in \mathcal{V}_u$, and each $\exists R.A \in \mathbf{L}$, we have $\langle w, z, \exists R.A \rangle \in \mathcal{E}'$ if and only if

$$\langle v, u, \exists R.A \rangle \in \mathcal{E}, \qquad \delta_{v,u}^{\exists R.A} \cap \mathrm{uknw}(v) \subseteq \mathcal{L}_v(w), \qquad \textit{and} \qquad \varrho_{v,u}^{\exists R.A} \cap \mathrm{uknw}(u) \subseteq \mathcal{L}_u(z).$$

(R8) For each context $v \in \mathcal{V}$, each vertex $w \in \mathcal{V}_v$, and all literals $L_1, L_2 \in \mathbf{L}$, we have $L_1 \prec'_w L_2$ if and only if $L_1 \prec_v L_2$ and no ϵ-edge $\langle u, w, \epsilon \rangle \in \mathcal{E}'$ exists such that $L_2 \in \mathrm{poss}'(u) \cap \mathrm{poss}'(w)$.

(R9) For each query $q = K \sqsubseteq M \in \mathcal{Q}$, context $v = \vartheta(q)$, and context $w = \vartheta'(q)$, we have $w \in \mathcal{V}_v$ and $(K \cup M) \cap \mathrm{uknw}(v) \subseteq \mathcal{L}_v(w)$.

Definition 10.9 can be intuitively understood as follows. Each hypergraph $\mathcal{H}_v$ captures the structure of a propositional problem that must be solved at context v: condition (R1) ensures that, for each context v of $\mathcal{D}$, hypergraph $\mathcal{H}_v$ contains the "unknown part" of each clause $K \sqsubseteq M \in \mathcal{O}$ for which all literals are possible at v; furthermore, condition (R2) ensures that, for each edge $\langle v, u, \exists R.A \rangle$ of $\mathcal{D}$, hypergraphs $\mathcal{H}_v$ and $\mathcal{H}_u$ contain all unknown literals that might be needed to apply the Pred rule; and finally, condition (R3) ensures that, for each query $q \in \mathcal{Q}$ and $v = \vartheta(q)$, hypergraph $\mathcal{H}_v$ contains the "unknown part" of q. Conditions (R4)–(R9) then essentially capture the intuition that each context v of $\mathcal{D}$ is replaced by a tree decomposition $\mathcal{T}_v$ of $\mathcal{H}_v$. Please note that, for each edge $\langle v, u, \exists R.A \rangle$ of $\mathcal{D}$, by condition (R2) and property (T1) of tree decompositions, there exists at least one pair of w and z satisfying condition (R7). Furthermore, for each query $q \in \mathcal{Q}$ and $v = \vartheta(q)$, context $\vartheta'(q)$ can be an arbitrary vertex $w \in \mathcal{V}_v$ satisfying $(K \cup M) \cap \mathrm{uknw}(v) \subseteq \mathcal{L}_v(w)$; due to condition (R3) and property (T1) of tree decompositions, such w is guaranteed to exist. Finally, please note that decomposition $\mathcal{D}$ can admit many ϵ-refinements; however, by the following theorem, if $\mathcal{D}$ is sound and admissible, then each ϵ-refinement of $\mathcal{D}$ is sound and admissible too.

Theorem 10.10. *For each sound, admissible, and ϵ-free decomposition $\mathcal{D}$ of $\mathcal{O}$ and $\mathcal{Q}$, each ϵ-refinement $\mathcal{D}'$ of $\mathcal{D}$ is also a sound and admissible decomposition of $\mathcal{O}$ and $\mathcal{Q}$.*

Proof. We first show that $\mathcal{D}'$ is sound. Consider arbitrary contexts $w, z \in \mathcal{V}$, and let $v, u \in \mathcal{V}$ be

such that $w \in \mathcal{V}_v$ and $z \in \mathcal{V}_u$. By condition (R5), we have $\mathrm{core}'(w) = \mathrm{core}(v)$, $\mathrm{knw}'(w) = \mathrm{knw}(v)$, $\mathrm{core}'(z) = \mathrm{core}(u)$, and $\mathrm{knw}'(z) = \mathrm{knw}(u)$.

Consider an arbitrary $\exists R.A$-edge $\langle w, z, \exists R.A\rangle \in \mathcal{E}'$. By condition (R7), then $\mathcal{D}$ contains the edge $\langle v, u, \exists R.A\rangle \in \mathcal{E}$. Since $\mathcal{D}$ is sound, $\mathcal{O} \models \mathrm{core}(v) \sqcap \exists R.A \sqsubseteq \exists R.[\mathrm{core}(u) \sqcap A]$ holds; but then, $\mathcal{O} \models \mathrm{core}'(w) \sqcap \exists R.A \sqsubseteq \exists R.[\mathrm{core}'(z) \sqcap A]$ holds as well.

Consider an arbitrary ϵ-edge $\langle w, z, \epsilon\rangle \in \mathcal{E}'$. By condition (R6), we have $v = u$, which clearly implies $\mathrm{core}'(w) = \mathrm{core}'(z)$, as required.

Finally, since $\mathcal{D}$ is sound, we have $\mathcal{O} \models \mathrm{core}(v) \sqsubseteq A$ for each $A \in \mathrm{knw}(v)$, so $\mathcal{O} \models \mathrm{core}'(w) \sqsubseteq A$ for each $A \in \mathrm{knw}'(w)$ clearly holds as well.

We next show that $\mathcal{D}'$ is admissible. Clearly, $\mathcal{D}'$ refers only to the literals occurring in $\mathcal{O} \cup \mathcal{Q}$, so $\mathcal{D}'$ is a decomposition of $\mathcal{O}$ and $\mathcal{Q}$. Furthermore, for each context $v \in \mathcal{V}$ the following conditions are satisfied.

$$\mathrm{knw}'(\mathcal{V}_v) = \bigcup_{w \in \mathcal{V}_v} \mathrm{knw}'(w) = \mathrm{knw}(v) \tag{10.53}$$

$$\mathrm{poss}'(\mathcal{V}_v) = \bigcup_{w \in \mathcal{V}_v} \mathrm{poss}'(w) \subseteq \mathrm{poss}(v) \tag{10.54}$$

Finally, for each ϵ-component $\mathcal{W}$ of $\mathcal{D}'$, a context $v \in \mathcal{V}$ exists such that $\mathcal{W} = \mathcal{V}_v$; conversely, for each context $v \in \mathcal{V}$, set $\mathcal{V}_v$ is an ϵ-component of $\mathcal{D}$. We next check that $\mathcal{D}'$ satisfies all the admissibility conditions listed in Definition 10.3.

Epsilon conditions:

(Condition E1) This property follows immediately from (R6) and the fact that the edges in $\mathcal{E}_v$ are undirected.

(Condition E2) By (R6), for each ϵ-component $\mathcal{V}_v$ of $\mathcal{D}'$, we have that $\mathcal{D}'_{\mathcal{V}_v}$ is equal to the graph $\langle \mathcal{V}_v, \mathcal{E}_v\rangle$, and the latter, being a tree decomposition, is an undirected tree.

(Condition E3) Consider an arbitrary ϵ-component $\mathcal{V}_v$ of $\mathcal{D}'$ and a literal $L \in \mathrm{poss}'(\mathcal{V}_v)$, and let $\Gamma := \{w \in \mathcal{V}_v \mid L \in \mathrm{poss}'(w)\}$. We have $L \in \mathrm{poss}(v)$ by (10.54), so either $L \in \mathrm{knw}(v)$ or $L \in \mathrm{uknw}(v)$. In the former case, we have $L \in \mathrm{poss}'(w)$ for each $w \in \mathcal{V}_v$, so $\Gamma = \mathcal{V}_v$ and Γ is clearly ϵ-connected. In the latter case, we have $\Gamma = \{w \in \mathcal{V}_v \mid L \in \mathcal{L}_v(w)\}$, so Γ is ϵ-connected by property (T2) of tree decompositions.

Structural conditions:

(Condition S1) By (R5), for each $v \in \mathcal{V}$ and each $w \in \mathcal{V}_v$, we have $\mathrm{core}'(w) = \mathrm{core}(v)$ and $\mathrm{knw}'(w) = \mathrm{knw}(v) \subseteq \mathrm{poss}'(w)$. Furthermore, $\mathrm{core}(v) \subseteq \mathrm{knw}(v)$ holds since $\mathcal{D}$ satisfies (S1), so we have $\mathrm{core}'(w) \subseteq \mathrm{knw}'(w) \subseteq \mathrm{poss}'(w)$.

(Condition S2) Consider an arbitrary ϵ-component $\mathcal{V}_u$ of $\mathcal{D}'$, an arbitrary edge $\langle w, z, \exists R.A\rangle \in \mathcal{E}'$ such that $z \in \mathcal{V}_u$, and an arbitrary literal $\forall \mathsf{inv}(R).C \in \mathrm{poss}'(\mathcal{V}_u)$; furthermore, let $v \in \mathcal{V}$ be the context of $\mathcal{D}$ such that $w \in \mathcal{V}_v$. By condition (R7) and $\langle w, z, \exists R.A\rangle \in \mathcal{E}'$, we have

$$\langle v, u, \exists R.A\rangle \in \mathcal{E}, \qquad \delta_{v,u}^{\exists R.A} \cap \mathrm{uknw}(v) \subseteq \mathcal{L}_v(w), \qquad \text{and} \qquad \varrho_{v,u}^{\exists R.A} \cap \mathrm{uknw}(v) \subseteq \mathcal{L}_u(z). \tag{10.55}$$

Now, by property (10.54) and $\forall \mathsf{inv}(R).C \in \mathrm{poss}'(\mathcal{V}_u)$, we have $\forall \mathsf{inv}(R).C \in \mathrm{poss}(u)$; hence, by admissibility condition (S2) for $\mathcal{D}$, we have $C \in \mathrm{poss}(v)$. Hence $C \in \delta_{v,u}^{\exists R.A}$ and $\forall \mathsf{inv}(R).C \in \varrho_{v,u}^{\exists R.A}$; together with (10.55), these conditions imply (10.56), so $\mathcal{D}'$ clearly satisfies admissibility condition (S2).

$$C \in \mathcal{L}_v(w) \cup \mathrm{knw}(v) = \mathrm{poss}'(w) \qquad\qquad \forall \mathsf{inv}(R).C \in \mathcal{L}_u(z) \cup \mathrm{knw}(u) = \mathrm{poss}'(z) \tag{10.56}$$

(Condition S3) Consider an arbitrary ϵ-component $\mathcal{V}_v$ of $\mathcal{D}'$ and a literal $\exists R.A \in \mathrm{poss}'(\mathcal{V}_v)$. By property (10.54), we have $\exists R.A \in \mathrm{poss}(v)$, so, by admissibility condition (S3) for $\mathcal{D}$, an $\exists R.A$-edge $\langle v, u, \exists R.A\rangle \in \mathcal{E}$ exists satisfying (10.57).

$$\mathrm{poss}(u) \supseteq \{A\} \cup \{B \mid \forall R.B \in \mathrm{poss}(v)\} \tag{10.57}$$

Since $\mathcal{T}_v$ and $\mathcal{T}_u$ are tree decompositions of $\mathcal{H}_v$ and $\mathcal{H}_u$, respectively, by property (R2) of Definition 10.9 and property (T1) of tree decompositions, contexts $w \in \mathcal{V}_v$ and $z \in \mathcal{V}_u$ exist such that the following two properties hold.

$$\delta_{v,u}^{\exists R.A} \cap \mathrm{uknw}(v) \subseteq \mathcal{L}_v(w) \tag{10.58}$$

$$\delta_{v,u}^{\exists R.A} \cap \mathrm{uknw}(u) \subseteq \mathcal{L}_u(z) \tag{10.59}$$

But then, by condition (R7), we have $\langle w, z, \exists R.A\rangle \in \mathcal{E}'$. We next show that this edge satisfies condi-

tion (S3)—that is, that

(i) $\text{poss}'(z) \supseteq \{A\} \cup \{B \mid \forall R.B \in \text{poss}'(\mathcal{V}_v)\}$, and

(ii) $\text{poss}'(w) \supseteq \{\exists R.A\} \cup \{\forall R.B \mid \forall R.B \in \text{poss}'(\mathcal{V}_v)\}$.

By Definition 10.9 we have $\exists R.A \in \delta_{v,u}^{\exists R.A}$ and $A \in \varrho_{v,u}^{\exists R.A}$; furthermore, for each $\forall R.B \in \text{poss}'(\mathcal{V}_v)$, we have $\forall R.B \in \text{poss}(v)$ by (10.54), and $B \in \text{poss}(u)$ by (10.57), which imply $\forall R.B \in \delta_{v,u}^{\exists R.A}$ and $B \in \varrho_{v,u}^{\exists R.A}$. But then, by (10.58) and (10.59), we have

$$\{A\} \cup \{B \mid \forall R.B \in \text{poss}'(\mathcal{V}_v)\} \subseteq \mathcal{L}_u(z) \cup \text{knw}(u) = \text{poss}'(z) \qquad \text{which proves (i), and}$$

$$\{\exists R.A\} \cup \{\forall R.B \mid \forall R.B \in \text{poss}'(\mathcal{V}_v)\} \subseteq \mathcal{L}_v(w) \cup \text{knw}(w) = \text{poss}'(w) \qquad \text{which proves (ii).}$$

Ontology condition:

Consider an arbitrary ϵ-component $\mathcal{V}_v$ of $\mathcal{D}'$ and a clause $K \sqsubseteq M \in \mathcal{O}$ with $K \subseteq \text{poss}'(\mathcal{V}_v)$. By property (10.54), $K \subseteq \text{poss}'(\mathcal{V}_v)$ implies $K \subseteq \text{poss}(v)$, so by the ontology condition for $\mathcal{D}$ we have $M \subseteq \text{poss}(v)$. By condition (R1) and property (T1) of tree decompositions, a context $w \in \mathcal{V}_v$ exists such that $(K \cup M) \cap \text{uknw}(v) \subseteq \mathcal{L}_v(w)$. But then, $K \cup M \subseteq \mathcal{L}_v(w) \cup \text{knw}(v) = \text{poss}'(w)$ holds, so $\mathcal{D}'$ satisfies the ontology condition.

Ordering conditions:

(Condition P1) Consider an arbitrary $\exists R.A$-edge $\langle w, z, \exists R.A\rangle \in \mathcal{E}'$. By condition (R7), then $v, u \in \mathcal{V}$ exist such that $w \in \mathcal{V}_v$, $z \in \mathcal{V}_u$, and $\langle v, u, \exists R.A\rangle \in \mathcal{E}$. Since $\mathcal{D}$ satisfies admissibility condition (P1), each literal $\forall \mathsf{inv}(R).C \in \mathbf{L}$ is $\prec_u$-minimal. But then, by condition (R8), each such $\forall \mathsf{inv}(R).C$ is also $\prec'_z$-minimal; hence, $\prec'_z$ is R-admissible.

(Condition P2) For each ϵ-edge $\langle u, w, \epsilon\rangle \in \mathcal{E}'$, all literals in $\text{poss}'(u) \cap \text{poss}'(w)$ are $\prec'_w$-minimal by condition (R8).

Covering condition:

Consider an arbitrary query $q = K \sqsubseteq M \in \mathcal{Q}$, and let $v = \vartheta(q)$ and $w = \vartheta'(q)$. By the covering condition for $\mathcal{D}$, context v covers $K \sqsubseteq M$, so $\text{core}(v) \subseteq K \subseteq \text{poss}(v)$ and M is $\prec_v$-minimal. By condition (R9), we have $w \in \mathcal{V}_v$ and $(K \cup M) \cap \text{uknw}(v) \subseteq \mathcal{L}_v(w)$. This and condition (R5) imply that

$$\text{core}'(w) = \text{core}(v) \subseteq K \subseteq \text{knw}(v) \cup \mathcal{L}_v(w) = \text{poss}'(w).$$

Furthermore, by (10.54) and condition (R5), we have

$$\begin{aligned} M \cap \mathrm{poss}'(\mathcal{V}_v) \subseteq M \cap \mathrm{poss}(v) &= M \cap [\mathrm{knw}(v) \cup \mathrm{uknw}(v)] \\ &\subseteq \mathrm{knw}(v) \cup [M \cap \mathrm{uknw}(v)] \subseteq \mathrm{knw}(v) \cup \mathcal{L}_v(w) = \mathrm{poss}'(w). \end{aligned}$$

Finally, by condition (R8), disjunction M is $\prec'_w$-minimal because M is $\prec_v$-minimal. Thus, context w covers $K \sqsubseteq M$ in $\mathcal{D}'$. □

Definition 10.9 can be straightforwardly turned into an algorithm for computing an ϵ-refinement $\mathcal{D}'$ of $\mathcal{D}$: first, we compute all hypergraphs satisfying conditions (R1)–(R3); second, we compute a tree decomposition $\mathcal{T}_v$ for each hypergraph $\mathcal{H}_v$; third, we construct decomposition $\mathcal{D}'$ so that conditions (R4)–(R9) are satisfied; and fourth, for each context v of $\mathcal{D}$ and each $w \in \mathcal{V}_v$, we define $\prec'_w$ by modifying $\prec_v$ to ensure condition (R8). As explained earlier, in condition (R9) we can don't-care nondeterministically choose $\vartheta(q)$, and at least one suitable context is guaranteed to exist. Such an algorithm is fixed-parameter tractable, as shown by the following theorem.

Theorem 10.11. *For every integer* mwd*, one can construct an ϵ-refinement $\mathcal{D}'$ of $\mathcal{D}$ satisfying* $\mathrm{wd}(\mathcal{D}') \leq \mathrm{mwd}$*, or determine that such $\mathcal{D}'$ does not exist, in time polynomial in $f(\mathrm{mwd})$, $\ln(\mathcal{D})$, and $\|\mathcal{O}\| + \|\mathcal{Q}\|$, for f a computable function.*

Proof. The **L**-hypergraphs $\mathcal{H}_v$ from conditions (R1)–(R3) of Definition (10.9) can clearly be constructed in time polynomial in $\ln(\mathcal{D})$ and $\|\mathcal{O}\| + \|\mathcal{Q}\|$.

An ϵ-refinement $\mathcal{D}'$ with $\mathrm{wd}(\mathcal{D}') \leq \mathrm{mwd}$ exists if and only if, for each $v \in \mathcal{V}$, hypertraph $\mathcal{H}_v$ admits a tree decomposition $\mathcal{T}_v$ with $\mathrm{wd}(\mathcal{T}_v) \leq \mathrm{mwd}$. For each $v \in \mathcal{V}$, all literals occurring in $\mathcal{N}_v$ are contained in $\mathcal{O} \cup \mathcal{Q}$, so by Lemma 10.1, computing one $\mathcal{T}_v$ with $\mathrm{wd}(\mathcal{T}_v) \leq \mathrm{mwd}$, or determining that one does not exist can be done in time $O(f(\mathrm{mwd}) \cdot (\|\mathcal{O}\| + \|\mathcal{Q}\|))$.

Decomposition $\mathcal{D}'$ can be obtained from $\mathcal{D}$ by simply following conditions (R4)–(R8). Furthermore, for each query $q \in \mathcal{Q}$, one can define $\vartheta'(q)$ as an arbitrary element of $\mathcal{V}_{\vartheta(q)}$ satisfying condition (R9); due to condition (R3) and property (T1) of tree decomposition, such an element is guaranteed to exist. This construction is polynomial in $\ln(\mathcal{D}')$ and $\|\mathcal{O}\| + \|\mathcal{Q}\|$. Since the length of each $\mathcal{D}_v$ is bounded by the time it takes to compute it, the length of $\mathcal{D}'$ is bounded by $O(\ln(\mathcal{D}) \cdot f(\mathrm{mwd}) \cdot (\|\mathcal{O}\| + \|\mathcal{Q}\|))$, so the entire construction runs in time polynomial in $f(\mathrm{mwd})$,

$\ln(\mathcal{D})$, and $\|\mathcal{O}\| + \|\mathcal{Q}\|$. □

This allows us to generalize our Theorem 9.16 on fixed-parameter tractability of subsumption reasoning from Section 9.4.

Theorem 10.12. *For each control $\mathcal{C}$, the following problem is fixed-parameter tractable:*

- Inputs: *a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$ and a set of queries $\mathcal{Q}$*
- Parameter: *an integer* mwd
- Problem: *return "yes" if an ϵ-refinement $\mathcal{D}$ of the $\mathcal{C}$-decomposition of $\mathcal{O}$ and $\mathcal{Q}$ exists with* $\mathrm{wd}(\mathcal{D}) \leq \mathrm{mwd}$*, and if also $\mathcal{O} \models K \sqsubseteq M$ holds for each query $K \sqsubseteq M \in \mathcal{Q}$*

Proof. Immediate by Theorems 9.11, 10.10, and 10.11, and Propositions 10.7 and 10.8. □

Beyond a critical point within a finite space, freedom diminishes as numbers increase. This is as true of humans as it is of gas molecules in a sealed flask.

—Frank Herbert
Dune

Chapter 11

Bounds on Decomposition Length

In this chapter we establish bounds on the sizes of decompositions of $\mathcal{O}$ and $\mathcal{Q}$. Towards this goal, in Section 11.1, we first present several soundness- and admissibility-preserving transformations for decompositions that can be used to eliminate redundant information. Then, for the upper bound (Section 11.2), we use these transformations to show that $\mathcal{O}$ and $\mathcal{Q}$ always admit a decomposition of minimal width and at most exponential length. For the lower bound (Section 11.3), we show that ontologies exist for which all decompositions of minimal width have exponential length.

11.1 Decomposition Transformations

In this section we present several soundness- and admissibility-preserving decomposition transformations that, under certain conditions, can delete edges and/or contexts from decompositions. These transformations can be useful in practice as they can reduce the number inferences of the consequence-based algorithm; moreover, we use these transformations in Section 11.2 to establish an exponential upper bound on decomposition size. Throughout this section, we fix a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$, a finite set of queries $\mathcal{Q}$, and a sound and admissible decomposition $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \mathrm{knw}, \mathrm{poss}, \prec, \vartheta \rangle$ of $\mathcal{O}$ and $\mathcal{Q}$.

Definition 11.1 introduces the notion of ϵ-edge contraction, which extends edge contraction from graph theory: given an edge $\langle v_1, v_2, \epsilon \rangle$ in $\mathcal{D}$, we merge contexts v_1 and v_2 into a new context v, eliminate the self-loop from v to v in order to preserve the tree shape of the enclosing ϵ-component, and define the core, known, and possible literals and the literal ordering of v to reflect both v_1 and v_2.

This transformation clearly preserves soundness and admissibility, but it can increase decomposition width. If, however, the ϵ-edge is redundant according to the criterion in Definition 11.1, then context v_1 is superfluous and so ϵ-edge contraction can only decrease or preserve decomposition width, but never increase it.

Definition 11.1. An ϵ-edge $\langle v_1, v_2, \epsilon \rangle \in \mathcal{E}$ is *redundant* in $\mathcal{D}$ if $\mathrm{uknw}(v_1) \subseteq \mathrm{poss}(v_2)$. The result of *contracting* a (not necessarily redundant) ϵ-edge $\langle v_1, v_2, \epsilon \rangle \in \mathcal{E}$ is obtained as follows.

1. Remove v_1 and v_2 from $\mathcal{V}$, and add a fresh context v to $\mathcal{V}$.
2. Replace each occurrence of v_1 and v_2 in ϑ and in $\mathcal{E}$ by v, and then remove $\langle v, v, \epsilon \rangle$ from $\mathcal{E}$.
3. Set $\mathrm{core}(v) \coloneqq \mathrm{core}(v_1)$, $\mathrm{knw}(v) \coloneqq \mathrm{knw}(v_1) \cup \mathrm{knw}(v_2)$, $\mathrm{poss}(v) \coloneqq \mathrm{poss}(v_1) \cup \mathrm{poss}(v_2)$, and $\prec_v \coloneqq \prec_{v_1} \cap \prec_{v_2}$.

Lemma 11.2. *Contracting an ϵ-edge preserves soundness and admissibility of a decomposition. Contracting a redundant ϵ-edge does not increase the width of a decomposition.*

Proof. It is straightforward to check that contracting an ϵ-edge preserves all conditions of Definitions 9.2 and 10.3. Assume now that a redundant ϵ-edge $\langle v_1, v_2, \epsilon \rangle \in \mathcal{E}$ is contracted. Then, $\mathrm{uknw}(v_1) \subseteq \mathrm{poss}(v_2)$ implies

$$\begin{aligned}\mathrm{uknw}(v) &= \mathrm{poss}(v) \setminus \mathrm{knw}(v) = [\mathrm{poss}(v_1) \cup \mathrm{poss}(v_2)] \setminus [\mathrm{knw}(v_1) \cup \mathrm{knw}(v_2)] \\ &\subseteq [[\mathrm{poss}(v_1) \setminus \mathrm{knw}(v_1)] \cup \mathrm{poss}(v_2)] \setminus \mathrm{knw}(v_2) = [\mathrm{uknw}(v_1) \cup \mathrm{poss}(v_2)] \setminus \mathrm{knw}(v_2) \\ &= \mathrm{poss}(v_2) \setminus \mathrm{knw}(v_2) = \mathrm{uknw}(v_2).\end{aligned}$$

Consequently, the width of a decomposition can only decrease as a result of contraction. □

Definition 11.3 specifies when a context w is broader than a context u in $\mathcal{D}$; roughly speaking, w is then "less specific" regarding core literals, but allows "more" possible literals. Given such w and u, we can then redirect each $\exists R.A$-edge ending at u so that the edge ends at w. That this operation preserves decomposition soundness and admissibility follows immediately from Lemma 11.4; moreover, this transformation manipulates only edges, so it clearly preserves decomposition width and length.

Definition 11.3. A context $w \in \mathcal{V}$ of $\mathcal{D}$ is *broader* than a context $u \in \mathcal{V}$ of $\mathcal{D}$ if $\mathrm{core}(w) \subseteq \mathrm{core}(u)$, $\mathrm{poss}(w) \supseteq \mathrm{poss}(u)$, $\prec_w \subseteq \prec_u$, and $\mathrm{poss}(\mathcal{W}) \subseteq \mathrm{poss}(\mathcal{U})$, where $\mathcal{W}$ and $\mathcal{U}$ are the ϵ-components of $\mathcal{D}$ such that $w \in \mathcal{W}$ and $u \in \mathcal{U}$. For such w and u, the result of *redirecting* u to w in $\mathcal{D}$ is obtained by replacing each edge $\langle v, u, \exists R.A\rangle$ in $\mathcal{E}$ with $\langle v, w, \exists R.A\rangle$, and by replacing u with w in ϑ.

Lemma 11.4. *Let $w, u \in \mathcal{V}$ be contexts of $\mathcal{D}$ such that w is broader than u. Then,*

(i) replacing an arbitrary $\exists R.A$-edge $\langle v, u, \exists R.A\rangle$ in $\mathcal{E}$ with $\langle v, w, \exists R.A\rangle$ preserves soundness and admissibility, and

(ii) each query q that is covered in u is also covered in w.

Proof. Claim (i): Assume that some $\langle v, u, \exists R.A\rangle$ in $\mathcal{E}$ is replaced with $\langle v, w, \exists R.A\rangle$. Only conditions (S2), (S3), and (P1) of Definition 10.3 concern $\exists R.A$-edges. To show that the "new" edge satisfies (S2), consider an arbitrary literal $\forall \mathsf{inv}(R).C \in \mathrm{poss}(\mathcal{W})$; then $\mathrm{poss}(\mathcal{W}) \subseteq \mathrm{poss}(\mathcal{U})$ implies $\forall \mathsf{inv}(R).C \in \mathrm{poss}(\mathcal{U})$; hence, by applying condition (S2) to the "old" edge $\langle v, u, \exists R.A\rangle$, we have $C \in \mathrm{poss}(v)$ and $\forall \mathsf{inv}(R).C \in \mathrm{poss}(u) \subseteq \mathrm{poss}(w)$, as required. Condition (S3) is clearly preserved due to $\mathrm{poss}(u) \subseteq \mathrm{poss}(w)$. Condition (P1) is preserved since $\prec_w \subseteq \prec_u$ and a subset of an R-admissible literal ordering is R-admissible.

Claim (ii): Consider an arbitrary query $K \sqsubseteq M$ that is covered in u; so, $\mathrm{core}(u) \subseteq K \subseteq \mathrm{poss}(u)$, $M \cap \mathrm{poss}(\mathcal{U}) \subseteq \mathrm{poss}(u)$, and M is $\prec_u$-minimal. Since context w is broader than context u, we have $\mathrm{core}(w) \subseteq K \subseteq \mathrm{poss}(w)$, $M \cap \mathrm{poss}(\mathcal{W}) \subseteq \mathrm{poss}(w)$, and M is $\prec_w$-minimal; but then, $K \sqsubseteq M$ is also covered in w. □

Definition 11.5 introduces a notion of redundancy for $\exists R.A$-edges. Intuitively, an $\exists R.A$-edge $\langle v, u, \exists R.A\rangle$ is redundant in $\mathcal{D}$ if either $\exists R.A$ is not possible in the ϵ-component of v so condition (S3) of Definition 10.3 is satisfied vacuously, or $\mathcal{D}$ contains another $\exists R.A$-edge that satisfies condition (S3) for $\exists R.A$. It is obvious that deleting a redundant $\exists R.A$-edge in $\mathcal{D}$ preserves soundness and admissibility; moreover, doing so can be particularly useful if $\mathcal{D}$ is an ϵ-refinement of an ϵ-free decomposition. In particular, for each $v, u \in \mathcal{V}$ in condition (R7) of Definition 10.9, decomposition $\mathcal{D}$ then contains an $\exists R.A$-edge from each $w \in \mathcal{V}_v$ to each $z \in \mathcal{V}_u$ that satisfy $\delta_{v,u}^{\exists R.A} \cap \mathrm{uknw}(v) \subseteq \mathcal{L}_v(w)$ and $\varrho_{v,u}^{\exists R.A} \cap \mathrm{uknw}(u) \subseteq \mathcal{L}_u(z)$; however, only one such edge may be needed to satisfy admissibility condition (S3), so deleting the redundant $\exists R.A$-edges might be beneficial in practice.

Definition 11.5. An $\exists R.A$-edge $\langle v, u, \exists R.A\rangle \in \mathcal{E}$ is *redundant* in $\mathcal{D}$ if either $\exists R.A \notin \text{poss}(\mathcal{W})$, or another $\exists R.A$-edge $\langle w, z, \exists R.A\rangle \in \mathcal{E}$ with $w \in \mathcal{W}$ exists that satisfies condition (S3) of Definition 10.3 for $\exists R.A \in \text{poss}(\mathcal{W})$, where $\mathcal{W}$ is the ϵ-component containing v.

Definition 11.6 says that an entire ϵ-component $\mathcal{U}$ of $\mathcal{D}$ may be redundant if, for each context $u \in \mathcal{U}$, either (i) there is no $\exists R.A$-edge from another ϵ-component to u, and u is not used to cover a query in $\mathcal{Q}$, or (ii) u can be redirected to a context in another ϵ-component. In such a case, we can redirect appropriately all contexts satisfying condition (ii) to obtain a decomposition in which no context in $\mathcal{U}$ has an incoming $\exists R.A$-edge from another ϵ-component; but then, we can clearly delete all contexts in $\mathcal{U}$ and all the related edges without affecting decomposition admissibility.

Definition 11.6. An ϵ-component $\mathcal{U}$ of $\mathcal{D}$ is *redundant* in $\mathcal{D}$ if at least one of the following holds for each context $u \in \mathcal{U}$:

- $v \in \mathcal{U}$ for each $\exists R.A$-edge $\langle v, u, \exists R.A\rangle \in \mathcal{E}$, and $\vartheta(q) \neq u$ for each query $q \in \mathcal{Q}$; or
- a context $w \in \mathcal{V} \setminus \mathcal{U}$ exists that is broader than u.

A result of *eliminating* such $\mathcal{U}$ from $\mathcal{D}$ is obtained from $\mathcal{D}$ as follows.

1. For each context $u \in \mathcal{U}$ such that either there exists an edge $\exists R.A$-edge $\langle v, u, \exists R.A\rangle \in \mathcal{E}$ with $v \notin \mathcal{U}$, or there exists a query $q \in \mathcal{Q}$ with $\vartheta(q) = u$, redirect u to some context $w \in \mathcal{V} \setminus \mathcal{U}$ that is broader than u.
2. Remove from the result of the previous step all contexts in $\mathcal{U}$ and all edges that are incident to a context in $\mathcal{U}$.

Lemma 11.7. *Eliminating a redundant ϵ-component preserves soundness and admissibility, and it does not increase the width.*

Proof. By Lemma 11.4, step 1 preserves soundness and admissibility. Furthermore, by the definition of when $\mathcal{U}$ is redundant in $\mathcal{D}$, after step 1 each context $u \in \mathcal{U}$ neither occurs in an $\exists R.A$-edge $\langle v, u, \exists R.A\rangle$ with $v \notin \mathcal{U}$, nor is it used to cover a query from $\mathcal{Q}$; hence, step 2 preserves decomposition admissibility. Furthermore, removal of edges and contexts clearly preserves decomposition soundness. Finally, steps 1 and 2 clearly do not increase the width of a decomposition. □

Please note that context redirection, elimination of $\exists R.A$-edges, and elimination of redundant ϵ-components also apply to context structures (with obvious modifications), and they can be applied during execution of Algorithm 8.10. In this way, we obtain "structural" redundancy elimination rules that can further improve the performance of the consequence-based algorithm.

11.2 Upper Bound

In this section we show that, for each ontology $\mathcal{O}$ and each set of queries $\mathcal{Q}$, a sound and admissible decomposition of $\mathcal{O}$ and $\mathcal{Q}$ exists that has minimum width and at most exponential length. Intuitively, this is because, given an arbitrary decomposition of minimal width, we can obtain the required decomposition as follows. First, we repeatedly eliminate redundant ϵ-edges; this ensures that each ϵ-component contains at most a polynomial number of contexts (cf. Lemma 11.8), so the number of distinct ϵ-components is exponential. Second, we eliminate "duplicates" by repeatedly eliminating redundant ϵ-components. We thus obtain a decomposition of $\mathcal{O}$ and $\mathcal{Q}$ of at most exponential length; since none of these transformations increases decomposition width, the resulting decomposition is of minimal width as well (cf. Theorem 11.9).

Lemma 11.8. *Let $\mathcal{O}$ be a normalized $\mathcal{ALCI}$ ontology, let $\mathcal{Q}$ be a finite set of queries, and let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \mathrm{knw}, \mathrm{poss}, \prec, \vartheta \rangle$ be an admissible decomposition of $\mathcal{O}$ and $\mathcal{Q}$ that has no redundant ϵ-edges. Then, for each ϵ-component $\mathcal{W}$ of $\mathcal{D}$, we have $|\mathcal{W}| \leq \max(|U|, 1)$ for $U = \bigcup_{w \in \mathcal{W}} \mathrm{uknw}(w)$.*

Proof. Let $\mathcal{W}$ be an arbitrary ϵ-component of $\mathcal{D}$ and let $n \coloneqq |\mathcal{W}|$. If there exists a context $v \in \mathcal{W}$ with $\mathrm{uknw}(v) = \emptyset$, since $\mathcal{D}$ has no redundant ϵ-edges, then v is the only context in $\mathcal{W}$, so $|\mathcal{W}| = 1$ and the lemma holds. Thus, in the rest of the proof, we assume that $\mathrm{uknw}(v) \neq \emptyset$ for each context $v \in \mathcal{W}$.

Let $v_1, \ldots, v_n$ be an arbitrary ordering of $\mathcal{W}$ obtained by an arbitrary depth-first traversal of the tree $\mathcal{D}_{\mathcal{W}}$. We define a function $\lambda \colon U \to \{1, \ldots, n\}$ by setting $\lambda(L) \coloneqq \min\{i \mid L \in \mathrm{uknw}(v_i)\}$. We claim that λ is surjective—that is, for each $1 \leq i \leq n$, a literal L exists such that $\lambda(L) = i$. For $i = 1$, since $\mathrm{uknw}(v_1) \neq \emptyset$, a literal $L \in \mathrm{uknw}(v_1)$ exists such that $\lambda(L) = 1$. Consider now an arbitrary i with $1 < i \leq n$, and let v_j be the parent of v_i in the depth-first traversal. Since $\mathcal{D}$ has no redundant ϵ-edges, we have $\mathrm{uknw}(v_i) \not\subseteq \mathrm{poss}(v_j)$, so a literal L exists such that $L \in \mathrm{uknw}(v_i) \setminus \mathrm{poss}(v_j)$. Now if $\lambda(L) < i$, then $v_{\lambda(L)}$ is visited in the depth-first traversal before v_i; thus, the unique path between

$v_{\lambda(L)}$ and v_i goes through v_j, so, by the connectedness condition (E3) of admissibility, we have $L \in \text{poss}(v_j)$; however, this contradicts our choice of L. Consequently, we have $\lambda(L) = i$, and so λ is surjective from U onto $\{1, \ldots, n\}$. But then, $|\mathcal{W}| = n \leq |U|$. □

Theorem 11.9. *Let $\mathcal{O}$ be a normalized $\mathcal{ALCI}$ ontology, let $\mathcal{Q}$ be a finite set of queries, and let* $\mathbf{L}$ *be the set of literals occurring in $\mathcal{O} \cup \mathcal{Q}$. For each sound and admissible decomposition $\mathcal{D}'$ of $\mathcal{O}$ and $\mathcal{Q}$, there exists a sound and admissible decomposition $\mathcal{D}$ of $\mathcal{O}$ and $\mathcal{Q}$ such that* $\text{wd}(\mathcal{D}) \leq \text{wd}(\mathcal{D}')$ *and* $\ln(\mathcal{D}) \leq 2^{O(|\mathbf{L}|^2)}$.

Proof. Let $\mathcal{D}'$ be an arbitrary sound and admissible decomposition of $\mathcal{O}$ and $\mathcal{Q}$, and let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \text{knw}, \text{poss}, \prec, \vartheta \rangle$ be an arbitrary decomposition obtained from $\mathcal{D}'$ by applying the following (nondeterministic) transformations.

1. Set $\prec_v := \emptyset$ for each context $v \in \mathcal{V}$.

2. While there are redundant ϵ-edges, pick one such ϵ-edge and contract it.

3. While there are redundant ϵ-components, pick one such ϵ-component and eliminate it.

By Lemmas 11.2 and 11.7, steps 2 and 3 preserve soundness and admissibility, and they do not increase the width; furthermore, step 1 trivially satisfies the two properties as well. To conclude our proof, we next show that $\ln(\mathcal{D}) \leq 2^{O(|\mathbf{L}|^2)}$.

Due to step 2, decomposition $\mathcal{D}$ contains no redundant ϵ-edges; hence, by Lemma 11.8, the cardinality of each ϵ-component of $\mathcal{D}$ is bounded by $|\mathbf{L}|$. We define the *type* of an ϵ-component $\mathcal{W}$ of $\mathcal{D}$ to be the pair $\langle \text{core}(\mathcal{W}), \{\text{poss}(v) \mid v \in \mathcal{W}\} \rangle$. Now assume that $\mathcal{U}$ and $\mathcal{W}$ are two different ϵ-components of $\mathcal{D}$ with the same type; then, for each context $u \in \mathcal{U}$, we can find a context $w \in \mathcal{W}$ with $\text{poss}(u) = \text{poss}(w)$; such w is broader than u because

- $\prec_w = \emptyset = \prec_u$ by step 1,
- $\text{core}(w) = \text{core}(\mathcal{W}) = \text{core}(\mathcal{U}) = \text{core}(u)$ by soundness condition (N2), and
- $\text{poss}(\mathcal{W}) = \bigcup_{v \in \mathcal{W}} \text{poss}(v) = \bigcup_{v \in \mathcal{U}} \text{poss}(v) = \text{poss}(\mathcal{U})$.

Consequently, such $\mathcal{U}$ is redundant in $\mathcal{D}$; however, due to step 3, decomposition $\mathcal{D}$ contains no redundant ϵ-components, so we obtain a contradiction. Therefore, all distinct ϵ-components of $\mathcal{D}$ are of a different type.

Thus, we can bound the number of ϵ-components of $\mathcal{D}$ by computing the number of possible types. For each ϵ-component $\mathcal{W}$ of $\mathcal{D}$, set core($\mathcal{W}$) is a subset of $\mathbf{L}$, so there are at most $2^{|\mathbf{L}|}$ such subsets; furthermore, set $\{\text{poss}(v) \mid v \in \mathcal{W}\}$ is a subset of $2^{\mathbf{L}}$ of cardinality at most $|\mathbf{L}|$, so the number of such subsets is bounded by

$$\sum_{k=0}^{|\mathbf{L}|} \binom{2^{|\mathbf{L}|}}{k} \leq \sum_{k=0}^{|\mathbf{L}|} (2^{|\mathbf{L}|})^k \leq (2^{|\mathbf{L}|})^{(|\mathbf{L}|+1)}.$$

Consequently, the number of ϵ-components in $\mathcal{D}$ is bounded by $2^{|\mathbf{L}|} \cdot (2^{|\mathbf{L}|})^{(|\mathbf{L}|+1)}$. Since each ϵ-component contains at most $|\mathbf{L}|$ contexts, we have $\ln(\mathcal{D}) \leq |\mathbf{L}| \cdot 2^{|\mathbf{L}|(|\mathbf{L}|+2)}$, as required. □

11.3 Lower Bound

In this section we show that a family of ontologies $\{\mathcal{O}_n\}$ and a fixed set of queries $\mathcal{Q}$ exist such that each sound and admissible decomposition of $\mathcal{O}_n$ and $\mathcal{Q}$ of minimum width necessarily has exponential length. Intuitively, this is because each ontology $\mathcal{O}_n$ has a "canonical" model $\mathcal{I}$ containing exponentially many domain elements, each of which satisfies a distinct combination of atomic concepts. Therefore, to obtain a decomposition of $\mathcal{O}_n$ and $\mathcal{Q}$ of minimal width, we must introduce a context for each domain element of $\mathcal{I}$, and thus such a decomposition will actually reflect the structure of $\mathcal{I}$. This is interesting because it shows that, in general, one cannot minimize decomposition width and still expect to obtain a practically useful decomposition (i.e., a decomposition of polynomial size). We use this observation as a justification for our practical approach to decomposition construction that we presented in Sections 9.4 and 10.4.

Theorem 11.10. *Let $\mathcal{Q} = \{C_0 \sqsubseteq \bot\}$. A family of $\mathcal{ALCI}$ ontologies $\{\mathcal{O}_n\}$ exists such that, for each $\mathcal{O}_n$,*

(i) a sound, admissible, and ϵ-free decomposition of $\mathcal{O}_n$ and $\mathcal{Q}$ of width 0 exists, and

(ii) each sound and admissible decomposition of $\mathcal{O}_n$ and $\mathcal{Q}$ of width 0 has length at least exponential in $\|\mathcal{O}_n\|$.

For readability, we split the proof of Theorem 11.10 into several claims. Let $\mathcal{Q} = \{C_0 \sqsubseteq \bot\}$, let n be a positive integer, and let $\mathcal{O}_n$ be the ontology (of size linear in n) containing the following

axioms for each $1 \leq i \leq n$.

$$C_{i-1} \sqsubseteq \exists R.C_i \qquad (11.1) \qquad\qquad C_{i-1} \sqsubseteq \exists S.C_i \qquad (11.5)$$

$$C_{i-1} \sqsubseteq \forall R.A_i \qquad (11.2) \qquad\qquad C_{i-1} \sqsubseteq \forall S.B_i \qquad (11.6)$$

$$A_i \sqsubseteq \forall R.A_i \qquad (11.3) \qquad\qquad A_i \sqsubseteq \forall S.A_i \qquad (11.7)$$

$$B_i \sqsubseteq \forall R.B_i \qquad (11.4) \qquad\qquad B_i \sqsubseteq \forall S.B_i \qquad (11.8)$$

Let $\mathbf{L} = \{C_0\} \cup \{C_i, A_i, B_i, \exists R.C_i, \exists S.C_i, \forall R.A_i, \forall S.A_i, \forall R.B_i, \forall S.B_i \mid 1 \leq i \leq n\}$ be the set of literals that occur in $\mathcal{O}_n$ and $\mathcal{Q}$. An *ABC-number* is a set of the form $X = \{X_1, \ldots, X_k, C_k\}$, where $0 \leq k \leq n$ and each X_i is either A_i or B_i. By a slight abuse of notation, we often treat X as the conjunction of the atomic concepts contained in X; furthermore, the *rank* of X is k; finally, X^R and X^S are ABC-numbers of rank $k + 1$ defined as

$$X^R \coloneqq \{X_1, \ldots, X_k, A_{k+1}, C_{k+1}\} \qquad \text{and} \qquad X^S \coloneqq \{X_1, \ldots, X_k, B_{k+1}, C_{k+1}\}.$$

Note that $\mathcal{O}_n \models X \sqsubseteq \exists R.X^R$ and $\mathcal{O}_n \models X \sqsubseteq \exists S.X^S$. Finally, for each ABC-number X, we define the set $\Gamma(X)$ of told subsumees of X w.r.t. $\mathcal{O}_n$ as follows:

$$\Gamma(X) \coloneqq X \cup \{L \in \mathbf{L} \mid \text{an atomic concept } D \in X \text{ exists such that } D \sqsubseteq L \in \mathcal{O}_n\}$$

One can readily check that $\mathcal{O}_n$ entails only told subsumptions—that is, for each ABC-number X and each $L \in \mathbf{L}$, we have

$$\mathcal{O}_n \models X \sqsubseteq L \text{ if and only if } L \in \Gamma(X); \qquad (11.9)$$

$$\mathcal{O}_n \models X \sqsubseteq \exists R.L \text{ if and only if } L \in \Gamma(X^R); \qquad (11.10)$$

$$\mathcal{O}_n \models X \sqsubseteq \exists S.L \text{ if and only if } L \in \Gamma(X^S). \qquad (11.11)$$

The following claim proves the first item of Theorem 11.10.

Claim 11.11. *There exists a sound, admissible, and ϵ-free decomposition $\mathcal{D}$ of $\mathcal{O}_n$ and $\mathcal{Q}$ such that* $\mathrm{wd}(\mathcal{D}) = 0$.

Proof. Let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \text{knw}, \text{poss}, \prec, \vartheta \rangle$ be defined as follows:

$$\begin{aligned}
\mathcal{V} &\coloneqq \{v_X \mid X \text{ is an ABC-number}\} \\
\text{core}(v_X) &\coloneqq X \\
\text{knw}(v_X) &\coloneqq \text{poss}(v_X) \coloneqq \Gamma(X) \\
\prec_{v_X} &\coloneqq \emptyset \\
\vartheta(C_0 \sqsubseteq \bot) &\coloneqq v_{\{C_0\}} \\
\mathcal{E} &\coloneqq \{\langle v_X, v_{X^R}, \exists R.C_{k+1}\rangle, \langle v_X, v_{X^S}, \exists S.C_{k+1}\rangle \mid \text{for each ABC-number } X \text{ of rank } k < n\}
\end{aligned}$$

Clearly, $\mathcal{D}$ is an ϵ-free decomposition of $\mathcal{O}_n$ and $\mathcal{Q}$. Furthermore, it is straightforward to check that $\mathcal{D}$ is sound. Finally, $\mathcal{D}$ satisfies all the admissibility conditions of Definition 9.3: conditions (S1) and (S2) hold trivially, the definition of $\mathcal{E}$ ensures that condition (S3) is satisfied, the definition of poss ensures that the ontology condition is satisfied, the trivial literal ordering $\prec_{v_X}$ always satisfies the ordering condition, and the query $C_0 \sqsubseteq \bot$ is covered in context $v_{\{C_0\}}$. Finally, since $\text{knw}(v_X) = \text{poss}(v_X)$ for each context $v_X \in \mathcal{V}$, we have $\text{wd}(\mathcal{D}) = 0$, as required. □

To prove the second item of Theorem 11.10, let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \text{knw}, \text{poss}, \prec, \vartheta \rangle$ be an arbitrary sound and admissible decomposition of $\mathcal{O}_n$ and $\mathcal{Q}$ such that $\text{wd}(\mathcal{D}) = 0$. Every ϵ-edge in a decomposition of width 0 is redundant in the sense of Definition 11.1, so we can assume without loss of generality that $\mathcal{D}$ is ϵ-free. We prove in Claim 11.13 that, for each ABC-number X, there exists a context $v_X \in \mathcal{V}$ such that $\text{poss}(v_X) = \Gamma(X)$. Since $\Gamma(X)$ is distinct for each ABC-number X, that $\mathcal{V}$ clearly contains at least 2^n contexts, which concludes the proof of Theorem 11.10. Towards proving Claim 11.13, we first prove the following auxiliary claim.

Claim 11.12. *Let $v \in \mathcal{V}$ be a context and X an ABC-number. If* $\text{core}(v) \subseteq \Gamma(X)$ *and* $\text{poss}(v) \supseteq X$, *then* $\text{poss}(v) = \Gamma(X)$.

Proof. Let $v \in \mathcal{V}$ be a context and X an ABC-number such that $\text{core}(v) \subseteq \Gamma(X)$ and $\text{poss}(v) \supseteq X$. By the ontology condition of Definition 9.3, set $\text{poss}(v)$ is closed under Γ; thus, $\text{poss}(v) \supseteq X$ implies $\text{poss}(v) \supseteq \Gamma(X)$. To show that $\text{poss}(v) \subseteq \Gamma(X)$ holds as well, consider an arbitrary literal $L \in \text{poss}(v)$. Since $\text{wd}(\mathcal{D}) = 0$, we have $L \in \text{knw}(v)$. But then, by soundness condition (N3) of Definition 9.2,

we have $\mathcal{O}_n \models \text{core}(v) \sqsubseteq L$. Furthermore, we have $\mathcal{O}_n \models X \sqsubseteq \Gamma(X)$ by (11.9), and $\text{core}(v) \subseteq \Gamma(X)$ clearly implies $\mathcal{O}_n \models \Gamma(X) \sqsubseteq \text{core}(v)$. Together, all these observations imply that $\mathcal{O}_n \models X \sqsubseteq L$ holds, which, by (11.9), implies $L \in \Gamma(X)$. Consequently, we have $\text{poss}(v) = \Gamma(X)$, as required. □

Claim 11.13. *For each ABC-number X, a context $v_X \in \mathcal{V}$ exists such that* $\text{poss}(v_X) = \Gamma(X)$.

Proof. The proof is by induction on the rank of X. For the base case, we have $X = \{C_0\}$. Since $X \sqsubseteq \bot \in \mathcal{Q}$, query $X \sqsubseteq \bot$ is covered in the context $v_X = \vartheta(X \sqsubseteq \bot)$; hence $\text{core}(v_X) \subseteq X \subseteq \text{poss}(v_X)$ by the definition of covering. But then, Claim 11.12 implies that $\text{poss}(v_X) = \Gamma(X)$, as required.

For the induction step, assume that the claim holds for each ABC-number of rank $k-1$, and consider an arbitrary ABC-number of the form $Y = \{X_1, \ldots, X_{k-1}, X_k, C_k\}$. Let $X = \{X_1, \ldots, X_{k-1}, C_{k-1}\}$. By the induction hypothesis, a context $v_X \in \mathcal{V}$ exists such that $\text{poss}(v_X) = \Gamma(X)$. We now consider the two possible forms of X_k.

Assume that $X_k = A_k$, which implies $Y = X^R$. By applying admissibility condition (S3) of Definition 9.3 to the existential restriction $\exists R.C_k \in \Gamma(X) = \text{poss}(v_X)$, an edge $\langle v_X, v_Y, \exists R.C_k \rangle \in \mathcal{E}$ exists such that

$$\text{poss}(v_Y) \supseteq \{C_k\} \cup \{D \mid \forall R.D \in \text{poss}(v_X)\} = X^R.$$

Soundness condition (N1) of Definition 9.2 implies $\mathcal{O}_n \models \text{core}(v_X) \sqcap \exists R.C_k \sqsubseteq \exists R.[\text{core}(v_Y) \sqcap C_k]$. Furthermore, we have $\mathcal{O}_n \models X \sqsubseteq \Gamma(X)$ by (11.9), and $[\text{core}(v_X) \sqcap \exists R.C_k] \subseteq \text{poss}(v_X) = \Gamma(X)$, which holds by the assumption, clearly implies $\mathcal{O}_n \models \Gamma(X) \sqsubseteq [\text{core}(v_X) \sqcap \exists R.C_k]$. Together, all these observations imply $\mathcal{O}_n \models X \sqsubseteq \exists R.[\text{core}(v_Y) \sqcap C_k]$; but then, $\text{core}(v_Y) \subseteq \Gamma(X^R)$ holds by (11.10). Thus, for $Y = X^R$, we have $\text{core}(v_Y) \subseteq \Gamma(X^R)$ and $\text{poss}(v_Y) \supseteq X^R$, so $\text{poss}(v_Y) = \Gamma(X^R) = \Gamma(Y)$ by Claim 11.12, as required.

The analysis for the case $X_k = B_k$ is analogous to the one above, with the difference that role S is used instead of role R. □

Any road followed precisely to its end leads precisely nowhere.

—Frank Herbert
Dune

Chapter 12

Discussion

In this chapter we discuss the practical value of our results by measuring decomposition width and length of realistic ontologies using several different expansion strategies (Section 12.1), and we conclude with suggestions for future work (Section 12.2).

12.1 Decomposition Width and Length in Practice

All ontologies used in this experiment were taken from the repository maintained by the Knowledge Representation and Reasoning Group at the University of Oxford.[1] The repository contains a diverse set of ontologies, including standard benchmark ontologies and numerous ontologies from the life sciences domain. To obtain our test corpus, we focused mostly on those ontologies in the repository that contain more than 1000 axioms and are not expressed in $\mathcal{EL}$ or DL-Lite$_{horn}$: ontologies expressed in one of these two languages have a decomposition of polynomial length and zero width, so we did not consider such ontologies interesting. However, we also included several smaller "toy" ontologies, such as Pizza and Wine, which were developed to demonstrate various ontology modeling constructs. In this way we obtained a corpus of 44 ontologies shown in Table 12.1. Since our framework is applicable only to normalized $\mathcal{ALCI}$ ontologies, we further transformed each ontology as follows. First, we eliminated from the ontology all axioms that are not supported in $\mathcal{SHI}$. Second, we transformed the result into a normalized $\mathcal{ALCI}$ ontology using the normalization procedure from Appendix B. For each test ontology, Table 12.1 shows (i) the number of

[1] An ontology with ID XXXXX can be downloaded from `http://www.cs.ox.ac.uk/isg/ontologies/UID/XXXXX.owl`, and a general description of the repository is available at `http://www.cs.ox.ac.uk/isg/ontologies/`.

terminological axioms in the original ontology, (ii) the number of atomic concepts, atomic roles, and axioms in the $\mathcal{SHI}$ ontology after elimination of unsupported axioms, and (iii) the number of literals, roles, and clauses in the final $\mathcal{ALCI}$ ontology after normalization; ontologies are grouped to match Table 12.2, and the rationale behind the grouping will be explained shortly. For each normalized ontology $\mathcal{O}$, we defined the set $\mathcal{Q}_\mathcal{O}$ to contain a query $A \sqsubseteq B$ for all pairs of atomic concepts A and B in $\mathcal{O}$ that were not introduced during normalization; thus, $\mathcal{Q}_\mathcal{O}$ contains all queries relevant for ontology classification.

We developed three controls, $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$, that use the cautious, cautious$_R$, and eager expansion strategies, respectively, described in Section 8.5.2. Each control $\mathcal{C}_i$ used mln = ∞—that is, we did not restrict the number of contexts introduced by the strategies. Although this is not formally allowed in our presentation of the algorithm, it allowed us to see how the strategies perform when run freely without being forced to reuse contexts. The maximum number of contexts that could be introduced by $\mathcal{C}_1$ and $\mathcal{C}_2$ was thus polynomial, but for $\mathcal{C}_3$ it was exponential in the ontology size. Finally, each $\mathcal{C}_i$ used an initialization function that, for each atomic concept A occurring in $\mathcal{Q}_\mathcal{O}$, introduces a single context v_A with core(v_A) = $\{A\}$ to cover all queries of the form $A \sqsubseteq B$. Please note that the expansion strategies of $\mathcal{C}_1$ and $\mathcal{C}_3$ can reuse these "initial" contexts, but the expansion strategy of $\mathcal{C}_2$ cannot since a context of the form v_A is distinct from the contexts of the form v_A^R (with an incoming role R) introduced by the cautious$_R$ expansion strategy.

For each normalized ontology $\mathcal{O}$, we conducted the following experiments. First, for each $1 \leq i \leq 3$, we computed the $\mathcal{C}_i$-decomposition $\mathcal{D}_i$ of $\mathcal{O}$ and $\mathcal{Q}_\mathcal{O}$ using the decomposition construction algorithm from Section 9.4. Second, for each $1 \leq i \leq 3$, we computed the width of an ϵ-refinement $\mathcal{D}_i'$ of $\mathcal{D}_i$ using the approach from Section 10.4. We computed tree decompositions of the relevant hypergraphs using the TreeD library.[2] Since our hypergraphs were large, we used a heuristic mode in which the library returns only an approximation of the treewidth, without the actual tree decomposition. Thus, we were able to establish only an upper bound on the width, but not the length of $\mathcal{D}_i'$; however, by Lemma 11.8, we know that the size of each ϵ-component of $\mathcal{D}_i'$ is at most wd($\mathcal{D}_i$), so an upper bound on ln($\mathcal{D}_i'$) is given by ln($\mathcal{D}_i$) · wd($\mathcal{D}_i$).

Table 12.2 summarizes the results of our experiments. For readability, the ontologies were grouped into three subtables: Table 12.2a contains those ontologies for which all three strategies

[2] http://www.itu.dk/people/sathi/treed/

Ontology	ID	Original logic	Original TBox	Reduced to $\mathcal{SHI}$ A	Reduced to $\mathcal{SHI}$ T	Reduced to $\mathcal{SHI}$ axioms	Normalized L	Normalized R	Normalized clauses
acgt	00001	$\mathcal{SROIQ}(\mathcal{D})$	5,457	1,750	247	4,981	3,851	417	9,030
fma-cons	00285	$\mathcal{ALCOIF}(\mathcal{D})$	123,090	41,646	139	116,270	85,576	243	121,833
go-anatomy-importer	00040	$\mathcal{SRIQ}$	130,480	58,193	209	130,411	177,934	356	279,928
nif-gross	00354	$\mathcal{SROIF}(\mathcal{D})$	6,630	4,042	62	6,580	6,126	79	9,493
uberon	00658	$\mathcal{SRIQ}$	23,644	7,838	82	23,575	42,960	118	77,373
envo-xp	00448	$\mathcal{SRIF}$	74,103	35,702	95	74,065	75,154	51	50,836
galen7	00792	$\mathcal{ALERIF}(\mathcal{D})$	45,260	28,447	964	44,531	341,544	1,610	370,519
go-xp-all	00483	$\mathcal{SRI}$	115,252	89,926	191	115,214	238,923	319	295,029
pato	00762	$\mathcal{SHIF}$	7,616	2,307	24	2,820	1,737	34	2,641
plant-trait-xp	00573	$\mathcal{SRIF}$	127,603	60,968	96	127,559	85,831	46	130,235
cdao	00410	$\mathcal{SROIQ}(\mathcal{D})$	3,175	890	106	3,056	2,934	192	6,152
dolce-all	00024	$\mathcal{SHOIN}(\mathcal{D})$	1,544	204	313	1,502	10,679	626	16,442
ero	00450	$\mathcal{SHOIF}(\mathcal{D})$	3,277	2,397	103	3,146	3,965	189	5,075
gardiner-wafa	00055	$\mathcal{SHIN}$	246	152	20	242	277	27	317
influenza-ontology	00507	$\mathcal{SROIN}(\mathcal{D})$	1,552	745	66	1,504	3,739	116	5,963
obi	00350	$\mathcal{SHOIN}(\mathcal{D})$	9,926	2,634	69	9,866	6,009	125	19,626
pizza	00793	$\mathcal{SHOIN}$	701	97	7	686	391	14	1,355
propreo	00772	$\mathcal{SHIN}$	565	474	30	548	994	48	1,086
sweet-space	00788	$\mathcal{SHOIN}(\mathcal{D})$	2,430	1,514	123	2,132	2,774	210	3,658
sweet-numerics	00789	$\mathcal{SHOIN}(\mathcal{D})$	2,499	1,499	137	2,185	2,768	239	3,599
sweet-phenomena	00790	$\mathcal{SHOIN}(\mathcal{D})$	2,658	1,721	112	2,367	2,871	191	3,734
fly-anatomy	00460	$\mathcal{SRI}$	19,477	7,798	21	19,466	10,391	27	19,534
galen-doctored	00029	$\mathcal{ALEHIF}^+$	4,762	2,748	413	4,586	11,447	534	13,769
galen-undoctored	00032	$\mathcal{ALEHIF}^+$	5,005	2,748	413	4,828	11,767	535	14,158
go	00764	$\mathcal{SRIF}$	76,286	36,495	13	76,279	82,191	22	123,506
worm-phenotype-xp	00675	$\mathcal{SHIF}$	90,522	51,413	113	90,497	74,594	95	100,966
aeo	00002	$\mathcal{SHIF}(\mathcal{D})$	20,317	759	47	3,372	1,487	92	5,598
lipid	00512	$\mathcal{ALCHIN}$	2,375	710	46	2,192	3,055	92	6,470
mammalian-phenotype-xp	00518	$\mathcal{SR}$	6,684	9,809	24	6,680	32,111	48	38,994
nci-thesaurus	00554	$\mathcal{SH}(\mathcal{D})$	127,739	91,226	123	127,729	157,186	244	226,658
nif-cell	00352	$\mathcal{SROIF}(\mathcal{D})$	3,482	2,770	61	3,431	3,212	62	4,366
pharmacogenomics	00566	$\mathcal{SHOIN}(\mathcal{D})$	52,813	45,000	201	52,616	48,056	279	53,797
protein	00791	$\mathcal{S}$	37,266	35,196	7	37,266	46,240	10	61,424
sao	00624	$\mathcal{SHIN}(\mathcal{D})$	2,802	764	36	2,608	1,231	58	4,595
sct-sep	00778	$\mathcal{SH}$	54,978	54,974	9	54,977	333,067	13	503,379
software	00636	$\mathcal{SHOIQ}(\mathcal{D})$	2,661	903	27	2,501	1,721	31	3,064
vaccine	00668	$\mathcal{SRIQ}$	12,516	5,410	137	12,470	12,412	201	20,730
wine	00783	$\mathcal{SHOIN}(\mathcal{D})$	395	98	17	159	302	30	388
bams-simplified	00004	$\mathcal{SHIF}$	18,822	1,110	12	18,818	4,413	14	19,575
iedm	00293	$\mathcal{ALUN}(\mathcal{D})$	4,310	395	888	1,877	4,821	1,771	4,532
molecular-function-xp	00533	$\mathcal{SRIF}$	84,408	42,165	82	84,368	73,981	20	106,124
neuron-ontology	00006	$\mathcal{ALEHI}^+$	1,260	1,119	25	1,245	1,419	1	1,208
sequence	00627	$\mathcal{SHI}$	2,812	2,136	24	3,029	3,179	26	4,074
world-fact-book	00104	$\mathcal{ALCHOI}(\mathcal{D})$	1,158	1,080	4	1,122	1,090	1	1,120

Table 12.1: Statistics for test ontologies

produce decompositions of different widths; Table 12.2b contains those ontologies for which the cautious and the cautious$_R$ strategies produce decompositions of equal widths; and Table 12.2c contains those ontologies for which the choice of a strategy does not affect decomposition width. Moreover, each subtable is split into two parts, where the bottom part contains ontologies for which $\mathrm{wd}(\mathcal{D}_3) = 0$: these are all Horn ontologies on which the decomposition construction algorithm with the eager strategy fully solves the reasoning problem. On the "go-anatomy-importer" ontology, strategies cautious and cautious$_R$ produce decompositions of very large width, which the TreeD library was unable to process. On the "nif-gross" ontology, the eager strategy seems to produce a decomposition of very large length, and our decomposition construction algorithm ran out of memory after introducing about five million contexts.

Our experiments show that decomposition length, even with the eager strategy, is in most cases commensurate with the number of atomic concepts. This is the main reason why consequence-based algorithms perform well in practice: our test ontologies do not seem to incur a substantial amount of and-branching; moreover, unlike the hypertableau algorithms, consequence-based algorithms do not suffer from unnecessary and-branching. The only outlier is the "nif-gross" ontology, on which the eager strategy produces very large decompositions; the cautious strategy, however, produces a decomposition of reasonably small width. This suggests that some sort of a hybrid strategy as discussed in Section 8.5.2 might be useful in practice.

The refined strategy cautious$_R$ does not seem to provide much practical benefit over the cautious strategy: there are only ten ontologies (all shown in Table 12.2a) on which $\mathrm{wd}(\mathcal{D}_1) > \mathrm{wd}(\mathcal{D}_2)$ holds, and in all cases the difference between $\mathrm{wd}(\mathcal{D}_1)$ and $\mathrm{wd}(\mathcal{D}_2)$ is negligible. Furthermore, only four decompositions satisfy $\mathrm{wd}(\mathcal{D}'_1) > \mathrm{wd}(\mathcal{D}'_2)$—that is, in all other cases, computing an ϵ-refinement of $\mathcal{D}_1$ and $\mathcal{D}_2$ produces decompositions of equal width. The cautious$_R$ strategy, however, can still be useful in practice: as we discussed in more detail in Section 8.5.2, contexts introduced by this strategy can have more refined literal orderings, but our framework cannot estimate the effects of a literal ordering on reasoning performance.

Interesting are also the results in the bottom part of Table 12.2c: all ontologies shown there contain universal restrictions and/or inverse roles, but they "behave" like $\mathcal{EL}$ ontologies in that there is no interaction between existential and universal restrictions and/or inverse roles; this is reflected by the fact that the cautious and the eager strategy introduce in most cases the same numbers of

Ontology	cautious $\ln(\mathcal{D}_1)$	cautious $\mathrm{wd}(\mathcal{D}_1)$	cautious $\mathrm{wd}(\mathcal{D}'_1)$	cautious$_R$ $\ln(\mathcal{D}_2)$	cautious$_R$ $\mathrm{wd}(\mathcal{D}_2)$	cautious$_R$ $\mathrm{wd}(\mathcal{D}'_2)$	eager $\ln(\mathcal{D}_3)$	eager $\mathrm{wd}(\mathcal{D}_3)$	eager $\mathrm{wd}(\mathcal{D}'_3)$
(a) $\mathrm{wd}(\mathcal{D}_1)$, $\mathrm{wd}(\mathcal{D}_2)$, and $\mathrm{wd}(\mathcal{D}_3)$ are all different:									
acgt	1,754	235	11	1,883	151	9	1,851	162	9
fma-cons	41,646	418	22	80,606	389	22	84,647	387	22
go-anatomy-importer	58,192	12,909	?	79,533	12,582	?	94,915	491	28
nif-gross	4,069	686	178	4,653	589	152	$> 5 \cdot 10^6$	?	?
uberon	7,383	4,931	369	10,966	4,600	115	10,241	199	18
envo-xp	35,702	2	1	50,247	1	1	35,704	0	0
galen7	30,472	3,850	47	43,521	3,833	47	49,344	0	0
go-xp-all	89,926	1,456	76	107,797	1,404	49	92,801	0	0
pato	2,306	2	1	2,331	1	1	2,308	0	0
plant-trait-xp	60,968	2	1	84,271	1	1	60,973	0	0
(b) $\mathrm{wd}(\mathcal{D}_1) = \mathrm{wd}(\mathcal{D}_2) > \mathrm{wd}(\mathcal{D}_3)$ and $\mathrm{wd}(\mathcal{D}'_1) = \mathrm{wd}(\mathcal{D}'_2) \geq \mathrm{wd}(\mathcal{D}'_3)$:									
cdao	890	369	90	1,203	369	90	1,449	82	25
dolce-all	236	2,180	72	336	2,180	72	453	1,819	62
influenza-ontology	842	554	32	1,032	554	32	1,060	237	12
obi	2,826	293	39	3,249	293	39	3,287	233	17
pizza	101	101	34	147	101	34	392	41	10
propreo	478	147	22	483	147	22	561	75	13
sweet-space	1,517	60	6	1,522	60	6	1,540	38	5
ero	2,399	100	11	2,479	100	11	2,456	99	11
gardiner-wafa	158	34	6	169	34	6	1167	33	6
sweet-numerics	1,503	44	5	1,508	44	5	1,518	34	5
sweet-phenomena	1,723	44	5	1,728	44	5	1,739	43	5
fly-anatomy	7,798	5	2	10,356	5	2	7,873	0	0
galen-doctored	3,020	139	14	3,824	139	14	3,364	0	0
galen-undoctored	3,042	139	14	4,091	139	14	3,460	0	0
go	36,494	140	16	46,813	140	16	36,859	0	0
worm-phenotype-xp	51,413	6	2	66,320	6	2	51,421	0	0
(c) $\mathrm{wd}(\mathcal{D}_1) = \mathrm{wd}(\mathcal{D}_2) = \mathrm{wd}(\mathcal{D}_3)$ and $\mathrm{wd}(\mathcal{D}'_1) = \mathrm{wd}(\mathcal{D}'_2) = \mathrm{wd}(\mathcal{D}'_3)$:									
aeo	851	32	6	1,124	32	6	963	32	6
lipid	860	1,010	186	983	1,010	186	3,561	1,008	186
mammalian-phenotype-xp	9,809	16	5	13,668	16	5	9,809	16	5
nci-thesaurus	91,226	48	11	107,788	48	11	103,977	48	11
nif-cell	2,769	48	7	2,796	48	7	2,771	48	7
pharmacogenomics	45,015	321	26	46,844	321	26	45,348	321	26
protein	35,196	22	5	36,636	22	5	35,196	22	5
sao	769	108	9	843	108	9	856	108	9
sct-sep	54,973	2,602	278	76,545	2,602	278	54,973	2,602	278
software	1,120	9	5	1,422	9	5	1,234	9	5
vaccine	5,691	1,137	246	7,087	1,137	246	6,584	1,137	246
wine	121	122	27	122	122	27	698	122	27
bams-simplified	1,187	0	0	3,424	0	0	3,154	0	0
iedm	395	0	0	395	0	0	395	0	0
molecular-function-xp	42,165	0	0	60,140	0	0	42,165	0	0
neuron-ontology	1,119	0	0	1,419	0	0	1,119	0	0
sequence	2,136	0	0	2,446	0	0	2,136	0	0
world-fact-book	1,080	0	0	1,080	0	0	1,080	0	0

Table 12.2: Decomposition width and length for test ontologies

contexts. The only exception is the "bams-simplified" ontology on which there is some interaction between existential and universal restrictions; however, even in this case, the cautious strategy can produce a decomposition of zero width. To understand how this can happen, consider the ontology $\{A \sqsubseteq \exists R.B,\ A \sqsubseteq \forall R.C,\ B \sqsubseteq C\}$: with the cautious strategy we get a context v_B with $\mathrm{core}(v_B) = \{B\}$, whereas with the eager strategy we get a context $v_{\{B,C\}}$ with $\mathrm{core}(v_{\{B,C\}}) = \{B, C\}$; however, in the former case C is a known consequence of B, so introducing a context with a "suboptimal" core does not increase decomposition width.

Our experiments further show that computing an ϵ-refinement can substantially reduce decomposition width: $\mathrm{wd}(\mathcal{D}'_i)$ is substantially lower than $\mathrm{wd}(\mathcal{D}_i)$ in all cases except when $\mathrm{wd}(\mathcal{D}_i)$ is already very small. Moreover, the eager strategy seems to produce decompositions of least width: with the exception of four ontologies ("dolce-all", "lipid", "sct-sep", and "vaccine"), decomposition $\mathcal{D}'_3$ has width below 30, and often much less than 30.

In all cases, decomposition width is substantially smaller than the number of literals in the ontology, so our results provide a tighter estimate of the algorithm's complexity than that obtained from the usual complexity arguments. This is particularly true for Horn ontologies: decomposition length on such ontologies is manageable and decomposition width is zero, which explains why such ontologies are generally easier to reason with. On non-Horn ontologies, however, the upper complexity bound obtained using our results may still be very large; for example, for a decomposition of width 30, the number of clauses derived in each context is bounded by $4^{30} \approx 1.15 \cdot 10^{18}$—clearly, no algorithm deriving that many clauses can be deemed practical. This bound, however, only analyzes what could happen in the worst case and, similarly to propositional resolution, such worst cases are unlikely to occur in practice. We observed that decomposition widths of our test ontologies are mainly determined by universal restrictions involving inverse roles: in order to guarantee that all clauses needed to apply the Pred rule are derived, sets $\varrho_{v,u}^{\exists R.A}$ are needed in condition (R7) of Definition 10.9, and these sets can be large. In our experience, however, clauses with many literals of the form $\forall \mathsf{inv}(R).C$ are rarely derived during reasoning, which is why most ontologies can still be handled in practice. Therefore, a small decomposition width provides an indication, but not a definite guarantee, that reasoning with a particular ontology is easy. In contrast, decomposition length measures directly the number of contexts constructed in the consequence-based algorithm, so length is generally a good measure of ontology hardness.

12.2 Conclusions and Future Work

In Part III of this thesis we presented a very general framework for consequence-based reasoning in description logics $\mathcal{ALCI}$ and $\mathcal{SHI}$. Then, in Part IV, we used this framework to introduce the notion of decomposition that allows us to quantify the effects of and- and or-branching during reasoning. Finally, we presented what we believe to be the first results on fixed-parameter tractability of description logic reasoning. We see several theoretical and practical challenges for future work.

On the theoretical side, one can try to extend our results to expressive description logics with number restrictions and nominals and thus obtain a consequence-based algorithm that can handle all of OWL DL. Number restrictions have already been successfully integrated into resolution-based procedures so, while not trivial, incorporating them into consequence-based algorithms should be possible. We have already started to extend the consequence-based framework to $\mathcal{SHIQ}$ [119], but this work is still at a very early stage. In contrast, the combination of nominals, number restrictions, and inverse roles is notoriously difficult: the complexity of subsumption checking in $\mathcal{SHOIQ}$ rises to coNEXPTIME [131], and it is currently unclear whether and how this combination of constructs can be handled in a deterministic consequence-based framework.

On the practical side, our results suggest several possible improvements to existing consequence-based reasoners. First, context reuse may allow reasoners to handle ontologies on which the eager strategy introduces a large number of contexts. Second, redundancy elimination techniques may turn out to further reduce the search space. Third, although computing tree decompositions in advance may incur significant overhead, we believe that computing ϵ-refinements "on demand" for contexts with large widths has the potential to further reduce the number of clauses derived. It will be interesting to incorporate these ideas into a consequence-based reasoner and try to determine the extent to which they help in practice.

Appendices

Do not count what you have lost. Count only what you still have.

—Brian Herbert and Kevin J. Anderson
Dune: The Machine Crusade

Appendix A

ABoxes and Safe Nominals in $\mathcal{EL}^+_\bot$

Up to now we have only considered TBox reasoning. In this appendix we demonstrate how general $\mathcal{EL}^+_\bot$ reasoning involving assertional axioms can be reduced to reasoning with only terminological axioms. We also demonstrate that this reduction works for a restricted but quite commonly used patterns of nominals in OWL EL ontologies.

Recall that 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.

We 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 A.1. *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 A.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 A.1, $\mathcal{O} \models \alpha$. Since $\mathcal{I} \models \alpha$ for no interpretation $\mathcal{I}$, this is only possible if $\mathcal{O}$ is inconsistent. □

The converses of Corollary A.2 and Lemma A.1 do not hold in general, but they hold if the occurrence of nominals is restricted in the following way.

Definition A.3 (Nominal Safety)**.** An $\mathcal{ELO}_\bot^+$ concept C is *safe* if C does not contain atomic concepts of the form N_a and all occurrences of nominals in C are 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 $\exists R.\{a\}$.

The key property of safety is that if $\mathcal{O}$ is safe, then the canonical model for $N(\mathcal{O})$ interprets every initialized and satisfiable N_a with a singleton set $\{x_{N_a}\}$. We will use the single instance x_{N_a} of this set to define the interpretation of the individual a, and thus obtain a model of $\mathcal{O}$.

Lemma A.4. *Let $\mathcal{O}$ be a safe $\mathcal{ELO}_\bot^+$ ontology, let* Input *be a set of expressions of the form* $\mathsf{init}(N(C))$ *where C is an n-safe concept, let* Closure *be the closure of* Input *under the rules in Fig. 3.1 w.r.t. $N(\mathcal{O})$, and let $\mathcal{J}$ be the canonical model of $N(\mathcal{O})$ w.r.t.* Closure*. Then, for every N_a such that* $\mathsf{init}(N_a) \in$ Closure *and $N_a \sqsubseteq \bot \notin$* Closure*, we have ${N_a}^{\mathcal{J}} = \{x_{N_a}\}$.*

Proof. By induction over rule applications in Fig. 3.1, it is easy to show that Closure can contain only expressions of the form

- $N(\mathsf{init}(C))$ such that C is n-safe,
- $N_a \sqsubseteq N_a$,
- $N(C \sqsubseteq D)$ such that $C \sqsubseteq D$ is safe,
- $N(C \xrightarrow{R} D)$ such that both C and D are n-safe.

Consider an arbitrary N_a such that $\mathsf{init}(N_a) \in \mathsf{Closure}$ and $N_a \sqsubseteq \bot \notin \mathsf{Closure}$. By Corollary 3.6, we have $x_{N_a} \in {N_a}^{\mathcal{J}}$. On the other hand, by the definition of the canonical model, if $x_C \in {N_a}^{\mathcal{J}}$, then $C \sqsubseteq N_a \in \mathsf{Closure}$; from the above invariant, this is only possible if $C = N_a$. Hence ${N_a}^{\mathcal{J}} = \{x_{N_a}\}$. □

We are now ready to prove the converse of Corollary A.2 and Lemma A.1 for safe occurrences of nominals.

Theorem A.5. *Let $\mathcal{O}$ be a safe $\mathcal{ELO}^+_\bot$ ontology. 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 $C \sqsubseteq D$, if $\mathcal{O} \models C \sqsubseteq D$, then $N(\mathcal{O}) \models N(C \sqsubseteq D)$.*

Proof. Take $\mathsf{Input} = \{\mathsf{init}(N(C)) \mid C \text{ is an n-safe } \mathcal{ELO}^+_\bot \text{ concept}\}$ (note that this is an infinite set) and let $\mathsf{Closure}$ be the closure of Input under the rules in Fig. 3.1 w.r.t. $N(\mathcal{O})$.

For every individual a, since $\{a\}$ is n-safe, we have $\mathsf{init}(N_a) \in \mathsf{Input} \subseteq \mathsf{Closure}$ and, since $N(\mathcal{O}) \not\models N_a \sqsubseteq \bot$ by the assumption and the inference rules are sound, we have $N_a \sqsubseteq \bot \notin \mathsf{Closure}$. Therefore, the canonical model $\mathcal{J}$ of $N(\mathcal{O})$ w.r.t. $\mathsf{Closure}$ is well-defined and, for every individual a, we have ${N_a}^{\mathcal{J}} = \{x_{N_a}\}$ by Lemma A.4.

Let us define an interpretation $\mathcal{I}$ by setting $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$, $A^{\mathcal{I}} = A^{\mathcal{J}}$ for atomic concepts, $R^{\mathcal{I}} = R^{\mathcal{J}}$ for roles, and $a^{\mathcal{I}} = x_{N_a}$ for individuals. Since the transformation $N(\cdot)$ merely replaces each $\{a\}$ by N_a and we have $\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\} = \{x_{N_a}\} = {N_a}^{\mathcal{J}}$, for every $\mathcal{ELO}^+_\bot$ axiom β we have $\mathcal{I} \models \beta$ iff $\mathcal{J} \models \mathcal{N}(\beta)$. Thus, since $\mathcal{J}$ is a model of $N(\mathcal{O})$, $\mathcal{I}$ is a model of $\mathcal{O}$. In particular, $\mathcal{O}$ is consistent.

Finally, we consider any safe concept inclusion $C \sqsubseteq D$ and show that $\mathcal{O} \models C \sqsubseteq D$ implies $N(\mathcal{O}) \models N(C \sqsubseteq D)$. By the definition of safety, C is n-safe and D is safe. Note that, without loss of generality, it is enough to consider the case when D is atomic: once we prove the theorem for atomic concepts D, it is then trivial to extend it to arbitrary safe concepts D by passing to the (safe) ontology $\mathcal{O}' \coloneqq \mathcal{O} \cup \{D \equiv A_D\}$ for a fresh atomic concept A_D. Thus, we assume that C is n-safe and

D is atomic (not of the form N_a). For the sake of contradiction, assume that $\mathcal{O} \models C \sqsubseteq D$ and $N(\mathcal{O}) \not\models N(C \sqsubseteq D) = N(C) \sqsubseteq D$. Since the inference rules are sound, we have $N(C) \sqsubseteq D \notin \mathsf{Closure}$ and also $N(C) \sqsubseteq \bot \notin \mathsf{Closure}$. Furthermore, since C is n-safe, we have $\mathsf{init}(N(C)) \in \mathsf{Input} \subseteq \mathsf{Closure}$. Then, by Corollary 3.6, we have $x_{N(C)} \in N(C)^{\mathcal{J}}$ and, by the definition of $D^{\mathcal{J}}$, we have $x_{N(C)} \notin D^{\mathcal{J}}$. This shows that $\mathcal{J} \not\models N(C) \sqsubseteq D = N(C \sqsubseteq D)$. It then follows that $\mathcal{I} \not\models C \sqsubseteq D$ which, since $\mathcal{I}$ is a model of $\mathcal{O}$, contradicts the assumption $\mathcal{O} \models C \sqsubseteq D$. □

Remark A.6. 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 A.5 it is sufficient to test if $N(\mathcal{O}) \not\models N_a \sqsubseteq \bot$ only for the individuals a occurring in $\mathcal{O}$ or, if $\mathcal{O}$ contains no individuals, if $N(\mathcal{O}) \not\models \top \sqsubseteq \bot$.

Using Theorem 3.2 and Theorem A.5, we can now describe a 'one pass' ontology realization procedure for computing all entailed instances of atomic concepts occurring in 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. 3.1 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.

Finally, we demonstrate that Theorem A.5 fails if the use of nominals is not safe. Take, for example, the ontology $\mathcal{O} = \{A \sqsubseteq \{a\},\ B \sqsubseteq \{a\},\ A \sqsubseteq \exists R.B\}$. Then $\mathcal{O}$ is consistent and $\mathcal{O} \models A \sqsubseteq B$ since for every model $\mathcal{I}$ of $\mathcal{O}$ we have either $A^{\mathcal{I}} = \emptyset$ or $A^{\mathcal{I}} = B^{\mathcal{I}} = \{a^{\mathcal{I}}\}$. However, for $N(\mathcal{O}) = \{A \sqsubseteq N_a,\ B \sqsubseteq N_a,\ A \sqsubseteq \exists R.B\}$, we have $N(\mathcal{O}) \not\models A \sqsubseteq B = N(A \sqsubseteq B)$. Reasoning in $\mathcal{ELO}^+_\bot$ with unrestricted occurrences of nominals is still possible (and tractable), but the procedure is significantly more complicated as it requires multiple saturation phases and/or complex types of axioms [65, 70].

One learns from books and example only that certain things can be done. Actual learning requires that you do those things.

—Frank Herbert
Children of Dune

Appendix B

Normalization of $\mathcal{SHI}$ Ontologies

In this appendix we show how to transform an arbitrary $\mathcal{SHI}$ ontology $\mathcal{O}$ into a normalized (cf. Section 8.1) $\mathcal{ALCI}$ ontology $\mathcal{O}'$ such that $\mathcal{O}'$ entails the same consequences as $\mathcal{O}$ over the atomic concepts occurring in $\mathcal{O}$. To prove correctness of our transformation, we use the framework of conservative extensions [82].

Definition B.1. Let $\mathcal{O}$ and $\mathcal{O}'$ be $\mathcal{SHI}$ ontologies, and let **A** and **T** be the sets of atomic concepts and atomic roles that occur in $\mathcal{O}$.

- $\mathcal{O}'$ is *conservative* over $\mathcal{O}$ if for every model $\mathcal{I}$ of $\mathcal{O}$ a model $\mathcal{I}'$ of $\mathcal{O}'$ exists such that $\mathcal{I}$ and $\mathcal{I}'$ have the same domain and coincide on the interpretation of **A** and **T**, and vice versa.
- $\mathcal{O}'$ is *concept-conservative* over $\mathcal{O}$ if for every model $\mathcal{I}$ of $\mathcal{O}$ a model $\mathcal{I}'$ of $\mathcal{O}'$ exists such that $\mathcal{I}$ and $\mathcal{I}'$ have the same domain and coincide on the interpretation of **A**, and vice versa.

Note that, unlike more standard notions of conservative extensions, we do not require that $\mathcal{O}'$ contains all axioms from $\mathcal{O}$. It is well-known and easy to show that the above model-theoretic notion of conservativity (resp. concept-conservativity) implies that the two ontologies entail the same consequences over **A** and **T** (resp. over **A**).

Our normalization method consists of two steps. Firstly, given an arbitrary $\mathcal{SHI}$ ontology $\mathcal{O}$, we apply structural transformation to produce a normalized $\mathcal{SHI}$ ontology $\mathcal{O}'$ that is conservative over $\mathcal{O}$. Secondly, having the normalized $\mathcal{SHI}$ ontology $\mathcal{O}'$, we eliminate transitivity and role inclusion axioms to produce a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}''$ that is concept-conservative over

$\mathcal{O}'$. Since in our case $\mathcal{O}'$ contains at least all atomic concepts that occur in $\mathcal{O}$, it follows that $\mathcal{O}''$ is also concept-conservative over $\mathcal{O}$. We describe the two normalization steps in more detail in Propositions B.2 and B.3 below. Both of them are well-known in the DL community.

Proposition B.2. *For every $\mathcal{SHI}$ ontology $\mathcal{O}$ there exists a normalized $\mathcal{SHI}$ ontology $\mathcal{O}'$ that is conservative over $\mathcal{O}$. Moreover, the size of $\mathcal{O}'$ is linear in the size of $\mathcal{O}$ and $\mathcal{O}'$ can be computed from $\mathcal{O}$ in polynomial time.*

Proof. In our formulation of structural transformation we use the standard notions of positive and negative occurrences of concepts in concepts and axioms. For $\mathcal{SHI}$, these can be defined inductively as follows: C occurs positively in C. If C occurs positively (resp. negatively) in C', then C also occurs positively (resp. negatively) in $C' \sqcap D$, $D \sqcap C'$, $C' \sqcup D$, $D \sqcup C'$, $\exists R.C'$, $\forall R.C'$, and $D \sqsubseteq C'$, and C occurs negatively (resp. positively) in $\neg C'$ and $C' \sqsubseteq D$. Finally, C occurs positively (resp. negatively) in an ontology $\mathcal{O}$ if it occurs positively (resp. negatively) in some axiom in $\mathcal{O}$. Note that a concept can occur both positively and negatively in an ontology.

Let $\mathcal{O}$ be a $\mathcal{SHI}$ ontology. Introduce a fresh (not yet in $\mathcal{O}$) atomic concept $[C]$ for each concept C occurring in $\mathcal{O}$. The *structural transformation* of C, denoted by $st(C)$, is defined as follows:

$$st(A) := A \qquad st(C \sqcap D) := [C] \sqcap [D]$$

$$st(\top) := \top \qquad st(C \sqcup D) := [C] \sqcup [D]$$

$$st(\bot) := \bot \qquad st(\exists R.C) := \exists R.[C]$$

$$st(\neg C) := \neg[C] \qquad st(\forall R.C) := \forall R.[C]$$

The *structural transformation* of $\mathcal{O}$ is a new ontology $\mathcal{O}'$ containing the same transitivity and role inclusion axioms as $\mathcal{O}$ and

- $st(C) \sqsubseteq [C]$ for every concept C occurring negatively in $\mathcal{O}$,
- $[D] \sqsubseteq st(D)$ for every concept D occurring positively in $\mathcal{O}$,
- $[C] \sqsubseteq [D]$ for each concept inclusion $C \sqsubseteq D$ in $\mathcal{O}$.

Note that $\|\mathcal{O}'\|$ is linear in $\|\mathcal{O}\|$ and $\mathcal{O}'$ can be computed from $\mathcal{O}$ in polynomial time.

We now prove that $\mathcal{O}'$ is conservative over $\mathcal{O}$. For one direction, it is easy to see that every model $\mathcal{I}$ of $\mathcal{O}$ can be extended into a model $\mathcal{I}'$ of $\mathcal{O}'$ by interpreting the new concepts $[C]$ by $[C]^{\mathcal{I}'} := C^{\mathcal{I}}$. For the other direction, we consider an arbitrary model $\mathcal{I}$ of $\mathcal{O}'$ and show that $\mathcal{I}$ is already a model of $\mathcal{O}$. Towards this goal, it is easy to show by structural induction on concepts C and D that

(i) if C occurs negatively in $\mathcal{O}$, then $C^{\mathcal{I}} \subseteq [C]^{\mathcal{I}}$;

(ii) if D occurs positively in $\mathcal{O}$, then $[D]^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

Let now $C \sqsubseteq D$ be an arbitrary concept inclusion in $\mathcal{O}$. Then C occurs negatively in $\mathcal{O}$, D occurs positively in $\mathcal{O}$, and $[C] \sqsubseteq [D] \in \mathcal{O}'$. By (i), (ii), and since $\mathcal{I}$ is a model of $\mathcal{O}'$, we have $C^{\mathcal{I}} \subseteq [C]^{\mathcal{I}}$, $[D]^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and $[C]^{\mathcal{I}} \subseteq [D]^{\mathcal{I}}$. These three set inclusions imply $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, thus $\mathcal{I} \models C \sqsubseteq D$. Since $C \sqsubseteq D$ was arbitrary, $\mathcal{I}$ is model of $\mathcal{O}$. This proves that $\mathcal{O}'$ is conservative over $\mathcal{O}$.

Finally, note that $\mathcal{O}'$ is not yet fully normalized: its concept inclusions are of the form $st(C) \sqsubseteq [C]$, $[D] \sqsubseteq st(D)$, and $[C] \sqsubseteq [D]$, which includes axioms such as $[C \sqcap D] \sqsubseteq [C] \sqcap [D]$ that are not normal clauses. Fortunately, all such axioms can be easily rewritten into normal clauses as follows:

$$
\begin{aligned}
[\top] \sqsubseteq \top &\rightsquigarrow \text{this is a tautology and can be removed from } \mathcal{O}' \\
\bot \sqsubseteq [\bot] &\rightsquigarrow \text{this is a tautology and can be removed from } \mathcal{O}' \\
[C \sqcap D] \sqsubseteq [C] \sqcap [D] &\rightsquigarrow [C \sqcap D] \sqsubseteq [C] \text{ and } [C \sqcap D] \sqsubseteq [D] \\
[C] \sqcup [D] \sqsubseteq [C \sqcup D] &\rightsquigarrow [C] \sqsubseteq [C \sqcup D] \text{ and } [D] \sqsubseteq [C \sqcup D] \\
[\neg C] \sqsubseteq \neg [C] &\rightsquigarrow [\neg C] \sqcap [C] \sqsubseteq \bot \\
\neg [C] \sqsubseteq [\neg C] &\rightsquigarrow \top \sqsubseteq [\neg C] \sqcup [C] \\
\exists R.[C] \sqsubseteq [\exists R.C] &\rightsquigarrow [C] \sqsubseteq \forall \mathsf{inv}(R).[\exists R.C] \\
\forall R.[C] \sqsubseteq [\forall R.C] &\rightsquigarrow \top \sqsubseteq [\forall R.C] \sqcup \exists R.[\neg C] \text{ and } [\neg C] \sqcap [C] \sqsubseteq \bot
\end{aligned}
$$

□

Proposition B.3. *For every normalized $\mathcal{SHI}$ ontology $\mathcal{O}$ there exists a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}'$ that is concept-conservative over $\mathcal{O}$. Moreover, $\mathcal{O}'$ can be computed from $\mathcal{O}$ in polynomial time.*

Proof. A normalized $\mathcal{SHI}$ ontology $\mathcal{O}$ can be transformed into a normalized $\mathcal{ALCI}$ ontology by eliminating all role inclusions and role transitivity axioms. Towards this goal, let $\sqsubseteq_{\mathcal{O}}$ be the smallest

reflexive and transitive binary relation on roles such that $R \sqsubseteq_{\mathcal{O}} S$ and $\mathsf{inv}(R) \sqsubseteq_{\mathcal{O}} \mathsf{inv}(S)$ for each role inclusion $R \sqsubseteq S \in \mathcal{O}$; furthermore, for each role R and each atomic concept C occurring in $\mathcal{O}$, let $A_{R,C}$ and $B_{R,C}$ be fresh atomic concepts unique for R and C. The elimination proceeds in three steps: step 1 ensures that universal restrictions occur only in clauses of the form $A \sqsubseteq \forall R.C$, step 2 encodes transitivity axioms using the well-known "box pushing" method, and step 3 eliminates role inclusions by expanding the role hierarchy into universal restrictions. We capture these steps by defining three ontologies obtained by transforming $\mathcal{O}$ as follows:

1. For each clause $K \sqsubseteq M \in \mathcal{O}$, ontology $\mathcal{O}_1$ contains the clause obtained from $K \sqsubseteq M$ by replacing each literal $\forall R.C \in M$ with $A_{R,C}$; furthermore, for each literal $\forall R.C$ occurring in a clause in $\mathcal{O}$, ontology $\mathcal{O}_1$ contains the axiom $A_{R,C} \sqsubseteq \forall R.C$.

2. Ontology $\mathcal{O}_2$ contains each clause in $\mathcal{O}_1$; furthermore, for each clause $A \sqsubseteq \forall R.C \in \mathcal{O}_1$ and each role S such that $S \sqsubseteq_{\mathcal{O}} R$ and either $\mathsf{Tra}(S) \in \mathcal{O}$ or $\mathsf{Tra}(\mathsf{inv}(S)) \in \mathcal{O}$, ontology $\mathcal{O}_2$ contains clauses

$$A \sqsubseteq \forall S.B_{S,C} \qquad \text{and} \qquad B_{S,C} \sqsubseteq \forall S.B_{S,C} \qquad \text{and} \qquad B_{S,C} \sqsubseteq C.$$

3. Ontology $\mathcal{O}_3$ contains each clause in $\mathcal{O}_2$; furthermore, for each clause $A \sqsubseteq \forall R.C \in \mathcal{O}_2$ and each role S such that $S \sqsubseteq_{\mathcal{O}} R$, ontology $\mathcal{O}_3$ contains clause $A \sqsubseteq \forall S.C$.

The resulting ontology $\mathcal{O}' \coloneqq \mathcal{O}_3$ is a normalized $\mathcal{ALCI}$ ontology that is concept-conservative over $\mathcal{O}$ (for a proof see, e.g., [118]). Observe that step 1 is not needed in case the input ontology $\mathcal{O}$ was obtained by the method in Proposition B.2 since that already produces universal restrictions only in clauses of the form $A \sqsubseteq \forall R.C$. □

For all their computerized precision, thinking machines can be confused in many different ways.

—Brian Herbert and Kevin J. Anderson
Dune: The Machine Crusade

Appendix C

The Hypertableau Algorithm

Most OWL DL reasoners are currently based on variants of tableau algorithms. To relate our techniques to the state of the art in ontology reasoning, in this appendix we present a variant of the hypertableau algorithm by Motik et al. [94] for normalized (cf. Section 8.1) $\mathcal{ALCI}$ ontologies.

We assume the existence of a countably infinite set of *named individuals*. An *individual* is a finite string of the form $a.i_1 \dots i_n$ where a is a named individual and $i_1 \dots i_n$ is a possibly empty sequence of integers. An individual of the form $s.i$ is *unnamed*, and it is a *successor* of individual s; *predecessor* is the inverse of successor; and *ancestor* and *descendant* are transitive closures of predecessor and successor, respectively. An *ABox* $\mathcal{A}$ is a finite set of facts of the form $\bot$, $L(s)$, or $R(s,t)$, for L a literal, R a role, and s and t individuals; furthermore, $\mathsf{ind}(\mathcal{A})$ is the set of individuals occurring in $\mathcal{A}$.

Termination of the algorithm is ensured via *anywhere blocking*; for $\mathcal{ALCI}$, the *single anywhere* variant suffices. Let $\lhd$ be an arbitrary strict order (i.e., an irreflexive and transitive relation) on the set of all individuals compatible with the ancestor relation (i.e., $s \lhd t$ holds whenever s is an ancestor of t). The *label* of an individual s in an ABox $\mathcal{A}$ is defined as $\mathcal{L}_{\mathcal{A}}(s) = \{L \mid L(s) \in \mathcal{A}\}$. By induction on $\lhd$, each individual s occurring in $\mathcal{A}$ is assigned a status as follows:

- s is *directly blocked* by an individual s' in $\mathcal{A}$ if both s and s' are unnamed, no ancestor of s' is blocked, $\mathcal{L}_{\mathcal{A}}(s) = \mathcal{L}_{\mathcal{A}}(s')$, and $s' \lhd s$;
- s is *indirectly blocked* if s is unnamed and some ancestor of s is directly blocked; and
- s is *blocked* if it is directly or indirectly blocked.

Hyp-rule	If $A_1 \sqcap \ldots \sqcap A_m \sqsubseteq L_1 \sqcup \ldots \sqcup L_n \in \mathcal{O}$, and an individual $s \in \mathsf{ind}(\mathcal{A}_0)$ exists that is not indirectly blocked such that $A_i(s) \in \mathcal{A}_0$ for each $1 \leq i \leq m$ and $L_j(s) \notin \mathcal{A}_0$ for each $1 \leq j \leq n$, then $\mathcal{A}_1 \coloneqq \mathcal{A}_0 \cup \{\bot\}$ if $n = 0$; and $\mathcal{A}_j \coloneqq \mathcal{A}_0 \cup \{L_j(s)\}$ for $1 \leq j \leq n$ if $n > 0$.
$\exists$-rule	If an individual s exists that is not blocked such that $\exists R.A(s) \in \mathcal{A}_0$ and no individual t exists such that $\{R(s,t),\ A(t)\} \subseteq \mathcal{A}_0$, then $\mathcal{A}_1 \coloneqq \mathcal{A}_0 \cup \{R(s,u),\ A(u)\}$ for u a fresh successor of s.
$\forall^+$-rule	If individuals s and t exists that are not indirectly blocked such that $\{\forall R.B(s),\ R(s,t)\} \subseteq \mathcal{A}_0$ and $B(t) \notin \mathcal{A}_0$, then $\mathcal{A}_1 \coloneqq \mathcal{A}_0 \cup \{B(t)\}$.
$\forall^-$-rule	If individuals s and t exists that are not indirectly blocked such that $\{[\forall \mathsf{inv}(R).B](t),\ R(s,t)\} \subseteq \mathcal{A}_0$ and $B(s) \notin \mathcal{A}_0$, then $\mathcal{A}_1 \coloneqq \mathcal{A}_0 \cup \{B(s)\}$.

Table C.1: Derivation rules of the hypertableau algorithm

Given a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$ and an ABox $\mathcal{A}$ containing only named individuals, the hypertableau algorithm constructs a *derivation* $D = (V, E, \rho)$ for $\mathcal{O}$ and $\mathcal{A}$, where V and E are the vertices and edges of a finite tree, and ρ labels each vertex with an ABox. A vertex $v \in V$ is *closed* if $\bot \in \rho(v)$; otherwise, v is *open*. Labeling ρ must satisfy the following conditions:

- $\rho(v) = \mathcal{A}$ for v the root of the tree;
- vertex v is a leaf if v is closed or no derivation rule from Table C.1 is applicable to $\mathcal{A}_0 = \rho(v)$;
- in all other cases, the children $v_1, \ldots, v_n$ of vertex v are labeled by ABoxes $\mathcal{A}_1, \ldots, \mathcal{A}_n$ obtained by applying one (arbitrarily chosen) derivation rule from Table C.1 to $\mathcal{A}_0 = \rho(v)$.

The algorithm is sound and complete: for each derivation D, we have that $\mathcal{O} \cup \mathcal{A}$ is satisfiable if and only if D contains an open leaf. Thus, checking whether $\mathcal{O} \models A \sqsubseteq B$ holds can be solved using the hypertableau algorithm by constructing a derivation for $\mathcal{O} \cup \{C \sqsubseteq A,\ B \sqcap C \sqsubseteq \bot\}$ and $\{C(a)\}$ where C is a fresh atomic concept, and then checking whether each derivation leaf is closed.

Due to inverse roles, blocking is *dynamic*: an individual can change its blocking status more than once as the inference rules are applied, which is why individuals can be indirectly blocked. The algorithm runs in N2ExpTime: it nondeterministically constructs a tree of individuals that can have exponential depth and a linear branching factor. Hence, although the number of nonblocked and directly blocked individuals is at most exponential, the number of indirectly blocked individuals in the tree can be doubly exponential; Motik et al. [94] discuss these issues in more detail.

Most men go through life unchallenged, except at the final moment.

—Frank Herbert
God Emperor of Dune

Appendix D

Completeness Proofs

In this appendix we prove Theorems 8.5 and 10.4.

D.1 Proof of Theorem 8.5

Throughout Appendix D.1 we fix a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$, an admissible context structure $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathrm{core}, \prec \rangle$, and a clause system $\mathcal{S}$ for $\mathcal{D}$ such that no inference rule from Table 8.1 is applicable to $\mathcal{S}$; furthermore, we fix $\mathbf{L}$ to be the set of all literals that occur in $\mathcal{O}$, $\mathcal{D}$, or $\mathcal{S}$. We prove Theorem 8.5 by showing the following contrapositive claim: $\mathcal{O} \not\models K \sqsubseteq M$ holds for each query $K \sqsubseteq M$ for which a context $v \in \mathcal{V}$ exists such that

- v is complete for $K \sqsubseteq M$,
- $K \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ for each literal $L \in K$, and
- $K \sqsubseteq M \mathrel{\hat{\notin}} \mathcal{S}(v)$.

To this end, we construct a model $\mathcal{I}$ of $\mathcal{O}$ that refutes each query satisfying the above conditions. The construction is organized as follows. Instead of directly constructing a model, for simplicity we construct a *pre-model*—a graph-like structure with nodes labeled by sets of literals. We introduce this notion in Section D.1.1, and we explain how to convert a pre-model into a model. In Section D.1.2 we show how to construct a propositional interpretation satisfying a set of clauses closed under hyperresolution. In Section D.1.3 we prove certain properties of our inference rules

that will allow us to use the construction from Section D.1.2. Finally, in Section D.1.4 we use these propositional interpretations to construct the desired pre-model of $\mathcal{O}$.

D.1.1 Pre-Interpretations and Pre-Models

In this section we introduce the notions of a pre-interpretation and a pre-model of an ontology, which we use to simplify the presentation of the subsequent proofs.

Definition D.1. A set of literals J *satisfies* a clause $K \sqsubseteq M$, written $J \models K \sqsubseteq M$, if $K \subseteq J$ implies $M \cap J \neq \emptyset$; otherwise, J *refutes* $K \sqsubseteq M$, written $J \not\models K \sqsubseteq M$.

A *pre-interpretation* is a labeled graph $I = \langle \Delta, E, J \rangle$ where Δ is a nonempty set of nodes, $E \subseteq \Delta \times \Delta \times \Sigma_R$ is a set of role-labeled edges, and $J \colon \Delta \to 2^{\Sigma_L}$ is a labeling of nodes by sets of literals satisfying the following two conditions.

($I_\exists$) For each node $x \in \Delta$ and each literal $\exists R.A \in J(x)$, an edge $\langle x, y, R \rangle \in E$ exists such that $A \in J(y)$.

($I_\forall$) For each edge $\langle x, y, R \rangle \in E$ and all literals $\forall R.B$ and $\forall \mathsf{inv}(R).C$, we have that

- $\forall R.B \in J(x)$ implies $B \in J(y)$, and
- $\forall \mathsf{inv}(R).C \in J(y)$ implies $C \in J(x)$.

Pre-interpretation $I = \langle \Delta, E, J \rangle$ *satisfies* a clause $K \sqsubseteq M$, written $I \models K \sqsubseteq M$, if $J(x) \models K \sqsubseteq M$ for each node $x \in \Delta$; otherwise, I *refutes* $K \sqsubseteq M$, written $I \not\models K \sqsubseteq M$. Finally, I is a *pre-model* of $\mathcal{O}$ if I satisfies each (normal) clause in $\mathcal{O}$.

Given a pre-interpretation $I = \langle \Delta, E, J \rangle$, we can construct an interpretation $\mathcal{I}$ for all atomic concepts $A \in \Sigma_A$ and all atomic roles $T \in \Sigma_T$ as follows.

$$\Delta^{\mathcal{I}} \coloneqq \Delta \tag{D.1}$$

$$A^{\mathcal{I}} \coloneqq \{x \mid A \in J(x)\} \tag{D.2}$$

$$T^{\mathcal{I}} \coloneqq \{\langle x, y \rangle \mid \langle x, y, T \rangle \in E\} \cup \{\langle y, x \rangle \mid \langle x, y, T^- \rangle \in E\} \tag{D.3}$$

Interpretation $\mathcal{I}$ satisfies the following property for each literal $L \in \Sigma_L$ and each node $x \in \Delta$:

$$L \in J(x) \quad \text{implies} \quad x \in L^{\mathcal{I}}. \tag{D.4}$$

For atomic concepts, property (D.4) holds by the definition of $A^{\mathcal{I}}$; furthermore, for literals of the form $\exists R.A$ and $\forall R.A$, property (D.4) holds due to conditions ($I_\exists$) and ($I_\forall$), respectively. The converse of property (D.4), however, holds only for atomic concepts, which is why all clauses in $\mathcal{O}$ must have only atomic concepts in their antecedents, and all queries must have only atomic concepts in their consequents. We next show that the interpretation $\mathcal{I}$ obtained as specified above is indeed a model of $\mathcal{O}$.

Lemma D.2. *Let $I = \langle \Delta, E, J \rangle$ be a pre-interpretation, and let $\mathcal{I}$ be the interpretation obtained from I as specified in* (D.1)–(D.3)*. If I satisfies a normal clause α, then $\mathcal{I} \models \alpha$. Furthermore, if I refutes a query q, then $\mathcal{I} \not\models q$.*

Proof. Let $\bigsqcap_{i=1}^{m} A_i \sqsubseteq \bigsqcup_{j=1}^{n} L_j$ be an arbitrary normal clause that is satisfied in I, and consider an arbitrary node $x \in \Delta$ such that $x \in A_i^{\mathcal{I}}$ for each $1 \leq i \leq m$. By (D.2), we have $A_i \in J(x)$ for each $1 \leq i \leq m$; but then, since I satisfies the clause by the assumption, we have $L_j \in J(x)$ for some $1 \leq j \leq n$; hence, we have $x \in L_j^{\mathcal{I}}$ for some $1 \leq j \leq n$ by (D.4). Since this holds for arbitrary $x \in \Delta$, we have $\mathcal{I} \models \bigsqcap_{i=1}^{m} A_i \sqsubseteq \bigsqcup_{j=1}^{n} L_j$.

Let $\bigsqcap_{i=1}^{m} L_i \sqsubseteq \bigsqcup_{j=1}^{n} A_j$ be an arbitrary query that is refuted in I. There exists a node $x \in \Delta$ such that $L_i \in J(x)$ for each $1 \leq i \leq m$, and $A_j \notin J(x)$ for each $1 \leq j \leq n$. But then, $x \in L_i^{\mathcal{I}}$ for each $1 \leq i \leq m$ by (D.4), and $x \notin A_j^{\mathcal{I}}$ for each $1 \leq j \leq n$ by (D.2); hence $\mathcal{I} \not\models \bigsqcap_{i=1}^{m} L_i \sqsubseteq \bigsqcup_{j=1}^{n} A_j$. □

Corollary D.3. *If a pre-model I of $\mathcal{O}$ exists that refutes a query $K \sqsubseteq M$, then $\mathcal{O} \not\models K \sqsubseteq M$.*

D.1.2 Constructing Literal Interpretations

In this section we show how the sets of clauses $\mathcal{S}(v)$ and the corresponding literal orderings $\prec_v$ can be used to construct the sets $J(x)$ of a pre-model $I = \langle \Delta, E, J \rangle$ that satisfies $\mathcal{O}$ but refutes a query $K \sqsubseteq M$. Our construction is independent from the selected context v; therefore, in the first part of this section we consider just a single set of clauses $\mathcal{N}$.

Throughout Section D.1.2 we fix a literal ordering $\prec$, a set of clauses $\mathcal{N}$ over $\mathbf{L}$, and a clause $K \sqsubseteq M$ over $\mathbf{L}$ satisfying the following conditions:

$$M \text{ is } \prec\text{-minimal,} \tag{D.5}$$

$$K \sqsubseteq L \mathbin{\hat{\in}} \mathcal{N} \text{ for each literal } L \in K, \text{ and} \tag{D.6}$$

$$K \sqsubseteq M \mathbin{\hat{\notin}} \mathcal{N}. \tag{D.7}$$

Next, in Definition D.4 we introduce a notion that determines when set $\mathcal{N}$ contains "sufficiently many" hyperresolution consequences w.r.t. literal ordering $\prec$.

Definition D.4. Set $\mathcal{N}$ is *closed under $\prec$-hyperresolution with a clause $\bigsqcap_{i=1}^{n} L_i \sqsubseteq M'$* (not necessarily contained in $\mathcal{N}$) if, for each choice of n clauses $K_i \sqsubseteq M_i \sqcup L_i \in \mathcal{N}$, $1 \leq i \leq n$, each satisfying $L_i \not\prec M_i$, we have $\bigsqcap_{i=1}^{n} K_i \sqsubseteq M' \sqcup \bigsqcup_{i=1}^{n} M_i \mathbin{\hat{\in}} \mathcal{N}$.

The next lemma follows trivially from Definition D.4 and the notion of clause strengthening.

Lemma D.5. *Let α_1 and α_2 be arbitrary clauses such that α_1 is a strengthening of α_2. If $\mathcal{N}$ is closed under $\prec$-hyperresolution with α_1, then $\mathcal{N}$ is also closed under $\prec$-hyperresolution with α_2.*

For $\prec$, $\mathcal{N}$, and $K \sqsubseteq M$ fixed as specified above, we next construct a set of literals J that refutes $K \sqsubseteq M$, but that satisfies each clause $\bigsqcup_{i=1}^{n} L_i \sqsubseteq M'$ for which $\mathcal{N}$ is closed under $\prec$-hyperresolution. We achieve this by essentially adapting the standard techniques from resolution theorem proving [18] to our setting. Towards this goal, we proceed as follows: we first present the construction of J; in Lemma D.6 we prove certain properties of the productive clauses in $\mathcal{N}$; in Lemma D.7 we show that J satisfies each clause $K' \sqsubseteq M'$ for which a strengthening is contained in $\mathcal{N}$ and that satisfies $K' \subseteq K$; in Lemma D.8 we show that J refutes $K \sqsubseteq M$; and finally in Lemma D.9 we prove the desired result.

Let $L_1, L_2, \ldots, L_\ell$ be an arbitrary total ordering of $\mathbf{L}$ that extends $\prec$ such that the elements of M precede all the remaining literals from $\mathbf{L}$—that is, for all i and j, we have that

$$L_i \prec L_j \text{ implies } i < j, \text{ and} \tag{D.8}$$

$$i < j \text{ and } L_j \in M \text{ imply } L_i \in M. \tag{D.9}$$

Since M is $\prec$-minimal by (D.5) and set $\mathbf{L}$ is finite, at least one such total ordering exists. For each $0 \leq k \leq \ell$, let $\mathbf{L}_k \coloneqq \{L_1, \ldots, L_k\}$, and let J_k be an inductively defined set of literals such that $J_0 \coloneqq \emptyset$ and, for each $0 < k \leq \ell$, we have

$$J_k \coloneqq \begin{cases} J_{k-1} \cup \{L_k\} & \text{if there exists a clause } K' \sqsubseteq M' \sqcup L_k \in \mathcal{N} \text{ such that} \\ & \qquad K' \subseteq K,\ M' \cap J_{k-1} = \emptyset, \text{ and } M' \subseteq \mathbf{L}_{k-1}, \\ J_{k-1} & \text{otherwise.} \end{cases} \tag{D.10}$$

Let $J \coloneqq J_\ell$. Each set J that can be obtained by such a construction (which is nondeterministic due to the choice of the total ordering) is called a *literal interpretation* of $\mathbf{L}$ w.r.t. $\mathcal{N}$ and $\prec$ refuting $K \sqsubseteq M$. Each clause $K' \sqsubseteq M' \sqcup L_k \in \mathcal{N}$ that satisfies the first condition in (D.10) for some k is said to be a *productive clause*, and the clause is said to *produce* the literal L_k in J. Clearly, each literal in J is produced by at least one productive clause in $\mathcal{N}$.

Lemma D.6. *Each productive clause $K' \sqsubseteq M' \sqcup L_k \in \mathcal{N}$ has $K' \subseteq K$, $M' \cap J = \emptyset$, and $L_k \not\prec M'$.*

Proof. Let $K' \sqsubseteq M' \sqcup L_k \in \mathcal{N}$ be an arbitrary clause producing literal L_k in J for some integer k. By (D.10), we have $K' \subseteq K$, $M' \cap J_{k-1} = \emptyset$, and $M' \subseteq \mathbf{L}_{k-1}$. Clearly $J_{k-1} = J \cap \mathbf{L}_{k-1}$, so we have $M' \cap J = \emptyset$. Finally, by (D.8), we have $L_k \not\prec L$ for each $L \in \mathbf{L}_{k-1}$; hence, due to $M' \subseteq \mathbf{L}_{k-1}$, we have $L_k \not\prec M$. □

Lemma D.7. *For each clause $K' \sqsubseteq M'$ such that $K' \sqsubseteq M' \mathrel{\hat{\in}} \mathcal{N}$ and $K' \subseteq K$, we have $M' \cap J \neq \emptyset$.*

Proof. Let $K' \sqsubseteq M'$ be an arbitrary clause satisfying $K' \sqsubseteq M' \mathrel{\hat{\in}} \mathcal{N}$ and $K' \subseteq K$. Then, a clause $K_1 \sqsubseteq M_1 \in \mathcal{N}$ exists such that $K_1 \subseteq K' \subseteq K$ and $M_1 \subseteq M'$ hold. Clause $K_1 \sqsubseteq \bot$ is thus a strengthening of $K \sqsubseteq M$, but $K \sqsubseteq M \mathrel{\hat{\notin}} \mathcal{N}$ by assumption (D.7); thus, $M_1 \neq \emptyset$. Let k be the largest integer for which $L_k \in M'$ holds; hence, clause $K_1 \sqsubseteq M_1$ is of the form $K_1 \sqsubseteq M_2 \sqcup L_k$ where $M_2 \subseteq \mathbf{L}_{k-1}$. The claim of this lemma holds trivially if $M_2 \cap J \neq \emptyset$. Furthermore, if $M_2 \cap J = \emptyset$, then by (D.10) clause $K_1 \sqsubseteq M_2 \sqcup L_k$ produces $L_k \in J$, so we again have $M' \cap J \neq \emptyset$, as required. □

Lemma D.8. *We have $J \not\models K \sqsubseteq M$—that is, $K \subseteq J$ and $M \cap J = \emptyset$.*

Proof. ($K \subseteq J$) Consider an arbitrary literal $L \in K$. Then $K \sqsubseteq L \mathrel{\hat{\in}} \mathcal{N}$ by assumption (D.6), so a conjunction $K' \subseteq K$ exists such that $K' \sqsubseteq \bot \in \mathcal{N}$ or $K' \sqsubseteq L \in \mathcal{N}$. The former contradicts $K \sqsubseteq M \mathrel{\hat{\notin}} \mathcal{N}$,

which is assumption (D.7); hence, we have $K' \sqsubseteq L \in \mathcal{N}$, so by (D.10) this clause produces $L \in J$.

($M \cap J = \emptyset$) Assume that a literal $L_k \in M \cap J$ exists, and let $K' \sqsubseteq M' \sqcup L_k \in \mathcal{N}$ be a clause that produces $L_k \in J$. By (D.10), we have $M' \subseteq \mathbf{L}_{k-1}$ and $K' \subseteq K$; furthermore, by (D.9) and $L_k \in M$, we have $M' \cup \{L_k\} \subseteq M$. But then, $K' \sqsubseteq M' \sqcup L_k \in \mathcal{N}$ contradicts $K \sqsubseteq M \hat{\notin} \mathcal{N}$, which is assumption (D.7). □

Lemma D.9. *For each clause α such that $\mathcal{N}$ is closed under $\prec$-hyperresolution with α, we have $J \models \alpha$.*

Proof. Assume that $\alpha = \bigsqcap_{i=1}^{n} L_i \sqsubseteq M'$ and that $L_i \in J$ for each $1 \leq i \leq n$. For each $1 \leq i \leq n$, let $K_i \sqsubseteq M_i \sqcup L_i$ be some clause that produces L_i in $\mathcal{J}$; by Lemma D.6, $K_i \subseteq K$, $M_i \cap J = \emptyset$, and $L_i \not\prec M_i$. Since $\mathcal{N}$ is closed under $\prec$-hyperresolution with α, we have $\bigsqcap_{i=1}^{n} K_i \sqsubseteq M' \sqcup \bigsqcup_{i=1}^{n} M_i \hat{\in} \mathcal{N}$. By $(\bigsqcap_{i=1}^{n} K_i) \subseteq K$ and Lemma D.7, we have $(M' \sqcup \bigsqcup_{i=1}^{n} M_i) \cap J \neq \emptyset$. Finally, since $M_i \cap J = \emptyset$ holds for each $1 \leq i \leq n$, we have $M' \cap J \neq \emptyset$. Consequently, we have $J \models \alpha$, as required. □

D.1.3 Properties of the Consequence-Based Inference Rules

To apply the construction from Section D.1.2 to our clause system $\mathcal{S}$ and the context structure $\mathcal{D}$ (which were fixed at the beginning of Appendix D.1), we next prove the following two auxiliary lemmas about the consequence-based inference rules. Lemma D.10 shows that, for each context $v \in \mathcal{V}$, the set of literals obtained by applying the above construction to $\mathcal{S}(v)$ satisfies each clause in $\mathcal{O}$; the lemma follows trivially from Definition D.4 and the fact that the Hyper rule is not applicable to $\mathcal{S}$. Lemma D.11 shows a similar property for the clauses that participate in the Pred rule.

Lemma D.10. *For each context $v \in \mathcal{V}$ and each clause $K \sqsubseteq M \in \mathcal{O}$, set $\mathcal{S}(v)$ is closed under $\prec_v$-hyperresolution with $K \sqsubseteq M$.*

Lemma D.11. *For each edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ and each clause $\alpha = A \sqcap \bigsqcap_{i=1}^{n} B_i \sqsubseteq \bigsqcup_{j=1}^{m} \forall \mathsf{inv}(R).C_j$ such that $\alpha \hat{\in} \mathcal{S}(u)$, set $\mathcal{S}(v)$ is closed under $\prec_v$-hyperresolution with $\exists R.A \sqcap \bigsqcap_{i=1}^{n} \forall R.B_i \sqsubseteq \bigsqcup_{j=1}^{m} C_j$.*

Proof. Consider an arbitrary edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ and a clause $\alpha = A \sqcap \bigsqcap_{i=1}^{n} B_i \sqsubseteq \bigsqcup_{j=1}^{m} \forall \mathsf{inv}(R).C_j$ such that $\alpha \hat{\in} \mathcal{S}(u)$. By the definition of $\hat{\in}$, a subset $\{B'_i\}_{i=1}^{n'}$ of $\{B_i\}_{i=1}^{n}$ and a subset $\{C'_j\}_{j=1}^{m'}$ of $\{C_j\}_{j=1}^{m}$

exist such that

$$A \sqcap \bigsqcap_{i=1}^{n'} B'_i \sqsubseteq \bigsqcup_{j=1}^{m'} \forall \mathsf{inv}(R).C'_j \in \mathcal{S}(u) \qquad \text{or} \qquad \bigsqcap_{i=1}^{n'} B'_i \sqsubseteq \bigsqcup_{j=1}^{m'} \forall \mathsf{inv}(R).C'_j \in \mathcal{S}(u).$$

The Pred rule is not applicable to clause system $\mathcal{S}$; thus, for each clause $K_0 \sqsubseteq M_0 \sqcup \exists R.A \in \mathcal{S}(v)$ satisfying $\exists R.A \not\prec_v M_0$, and for each choice of n' clauses $K_i \sqsubseteq M_i \sqcup \forall R.B'_i$, $1 \leq i \leq n'$, satisfying $\forall R.B'_i \not\prec_v M_i$, we have

$$K_0 \sqcap \bigsqcap_{i=1}^{n'} K_i \sqsubseteq M_0 \sqcup \bigsqcup_{i=1}^{n'} M_i \sqcup \bigsqcup_{j=1}^{m'} C'_j \mathrel{\hat{\in}} \mathcal{S}(v).$$

Thus, $\mathcal{S}(v)$ is closed under $\prec_v$-hyperresolution with the clause $\exists R.A \sqcap \bigsqcap_{i=1}^{n'} \forall R.B'_i \sqsubseteq \bigsqcup_{j=1}^{m'} C'_j$. Then, by Lemma D.5, $\mathcal{S}(v)$ is also closed under $\prec_v$-hyperresolution with $\exists R.A \sqcap \bigsqcap_{i=1}^{n} \forall R.B_i \sqsubseteq \bigsqcup_{j=1}^{m} C_j$ since the former clause is a strengthening of the latter clause. □

D.1.4 The Completeness Claim

We now complete the proof of Theorem 8.5 by constructing a pre-interpretation that satisfies $\mathcal{O}$ but refutes the relevant queries. To simplify the presentation, given a context $v \in \mathcal{V}$ and a clause $K \sqsubseteq M$, we write $v \nvdash K \sqsubseteq M$ if and only if

- M is $\prec_v$-minimal (i.e., v is complete for $K \sqsubseteq M$),
- $K \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ for each literal $L \in K$, and
- $K \sqsubseteq M \mathrel{\hat{\notin}} \mathcal{S}(v)$.

Claim D.12. *For each context $v \in \mathcal{V}$ and each clause $K \sqsubseteq M$, if $K \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ for each $L \in K$ and $K \sqsubseteq M \mathrel{\hat{\notin}} \mathcal{S}(v)$, then $K \subseteq \mathbf{L}$.*

Proof. Assume that there exists a literal $L \in K \setminus \mathbf{L}$; then, L does not occur in $\mathcal{S}(v)$, so $K \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ implies $K \sqsubseteq \bot \mathrel{\hat{\in}} \mathcal{S}(v)$, which contradicts the assumption that $K \sqsubseteq M \mathrel{\hat{\notin}} \mathcal{S}(v)$. Hence $K \subseteq \mathbf{L}$. □

Theorem 8.5 holds vacuously if no context $v \in \mathcal{V}$ and no query $K \sqsubseteq M$ exist such that $v \nvdash K \sqsubseteq M$. Thus, we assume that $v \nvdash K \sqsubseteq M$ holds for at least one context $v \in \mathcal{V}$ and at least one query $K \sqsubseteq M$; by the definition of clause strengthening, we then also have $v \nvdash K \sqsubseteq M'$ for $M' = M \cap \mathbf{L}$; the latter clause is over $\mathbf{L}$ by Claim D.12. Let $I = \langle \Delta, E, J \rangle$ be defined as follows.

- Set Δ is the smallest set that contains a distinguished element $\delta^v_{K \sqsubseteq M}$ for each context $v \in \mathcal{V}$ and each clause $K \sqsubseteq M$ over $\mathbf{L}$ such that $v \nvdash K \sqsubseteq M$. Set Δ is not empty due to the above assumption.

- For each $\delta^v_{K \sqsubseteq M} \in \Delta$, let $J(\delta^v_{K \sqsubseteq M})$ be an arbitrary literal interpretation of $\mathbf{L}$ w.r.t. $\mathcal{S}(v)$ and $\prec_v$ refuting $K \sqsubseteq M$.

- Set E is the smallest set containing $\langle \delta^v_{K_1 \sqsubseteq M_1}, \delta^u_{A \sqcap K_2 \sqsubseteq M_2}, R \rangle$ for each element $\delta^v_{K_1 \sqsubseteq M_1} \in \Delta$, each literal $\exists R.A \in J(\delta^v_{K_1 \sqsubseteq M_1})$, and each edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ such that $u \nvdash A \sqcap K_2 \sqsubseteq M_2$, where K_2 and M_2 are as specified below.

$$K_2 = \bigsqcap \{B \mid \forall R.B \in J(\delta^v_{K_1 \sqsubseteq M_1})\} \tag{D.11}$$

$$M_2 = \bigsqcup \{\forall \mathsf{inv}(R).C \in \mathbf{L} \mid C \notin J(\delta^v_{K_1 \sqsubseteq M_1})\} \tag{D.12}$$

In the last item, we have $K_2 \cup M_2 \subseteq \mathbf{L}$ and $A \in \mathbf{L}$; thus, $u \nvdash A \sqcap K_2 \sqsubseteq M_2$ ensures $\delta^u_{A \sqcap K_2 \sqsubseteq M_2} \in \Delta$, and so E is correctly defined. Next, in Claim D.13 we show that I is a pre-interpretation, in Claim D.14 we show that I is a pre-model of $\mathcal{O}$, and in Claim D.15 we show that I refutes the relevant queries. The claim of Theorem 8.5 then follows from Claims D.14 and D.15, and Corollary D.3.

Claim D.13. *Structure I satisfies conditions $(I_\exists)$ and $(I_\forall)$ of pre-interpretations from Definition D.1.*

Proof. (Property $I_\exists$) Consider an arbitrary element $\delta^v_{K_1 \sqsubseteq M_1} \in \Delta$ and a literal $\exists R.A \in J(\delta^v_{K_1 \sqsubseteq M_1})$; we next show that an edge $\langle \delta^v_{K_1 \sqsubseteq M_1}, \gamma, R \rangle \in E$ exists such that $A \in J(\gamma)$. The Succ rule is not applicable to $\mathcal{S}$, so an edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ exists satisfying

$$L \sqsubseteq L \,\hat{\in}\, \mathcal{S}(u) \text{ for each } L \in \{A\} \cup \mathbf{B}_p, \text{ where } \mathbf{B}_p = \{B \mid K' \sqsubseteq M' \sqcup \forall R.B \in S(v) \text{ and } \forall R.B \not\prec_v M'\}. \tag{D.13}$$

Let K_2 and M_2 be as in (D.11) and (D.12). The following observations establish $u \nvdash A \sqcap K_2 \sqsubseteq M_2$.

- M_2 is $\prec_u$ -minimal by the ordering condition of Definition 9.3.

- Consider an arbitrary literal $L \in A \sqcap K_2$; we show that $A \sqcap K_2 \sqsubseteq L \,\hat{\in}\, \mathcal{S}(u)$. Assume that $L \in K_2$, so L is an atomic concept $L = B$. Then, by (D.11), we have $\forall R.B \in J(\delta^v_{K_1 \sqsubseteq M_1})$; thus,

there exists at least one clause $K' \sqsubseteq M' \sqcup \forall R.B \in S(v)$ with $\forall R.B \not\prec_v M'$ that produces $\forall R.B$ in $J(\delta^v_{K_1 \sqsubseteq M_1})$; hence $B \in \mathbf{B}_p$. But then $L \sqsubseteq L \mathbin{\hat{\in}} \mathcal{S}(u)$ holds by (D.13), which is stronger than the required $A \sqcap K_2 \sqsubseteq L \mathbin{\hat{\in}} \mathcal{S}(u)$. The case when $L = A$ follows directly from (D.13).

- Assume for contradiction that $A \sqcap K_2 \sqsubseteq M_2 \mathbin{\hat{\in}} \mathcal{S}(u)$. By Lemma D.11, set $\mathcal{S}(v)$ is closed under $\prec_v$-hyperresolution with clause

$$\exists R.A \sqcap \bigsqcap \{\forall R.B \mid \forall R.B \in J(\delta^v_{K_1 \sqsubseteq M_1})\} \sqsubseteq \bigsqcup \{C \mid \forall \mathsf{inv}(R).C \in \mathbf{L} \text{ and } C \notin J(\delta^v_{K_1 \sqsubseteq M_1})\}.$$

  Recall that $\exists R.A \in J(\delta^v_{K_1 \sqsubseteq M_1})$, so the above clause is clearly refuted in $J(\delta^v_{K_1 \sqsubseteq M_1})$. However, by Lemma D.9, the clause is satisfied in $J(\delta^v_{K_1 \sqsubseteq M_1})$, which is a contradiction. Consequently, we have $A \sqcap K_2 \sqsubseteq M_2 \mathbin{\hat{\notin}} \mathcal{S}(u)$, as required.

Thus, we have $u \nvdash A \sqcap K_2 \sqsubseteq M_2$, which implies $\langle \delta^v_{K_1 \sqsubseteq M_1}, \delta^u_{A \sqcap K_2 \sqsubseteq M_2}, R \rangle \in E$ by the definition of E. Finally, $A \in J(\delta^u_{A \sqcap K_2 \sqsubseteq M_2})$ holds by Lemma D.8. All of these observations prove that I satisfies property ($I_\exists$) of Definition D.1.

(Property $I_\forall$) Consider an arbitrary edge $\langle \delta^v_{K_1 \sqsubseteq M_1}, \delta^u_{A \sqcap K_2 \sqsubseteq M_2}, R \rangle \in E$ as in the definition of E, as well as arbitrary literals $\forall R.B$ and $\forall \mathsf{inv}(R).C$.

- Assume that $\forall R.B \in J(\delta^v_{K_1 \sqsubseteq M_1})$. Then $B \in K_2$ holds by (D.11); but then, Lemma D.8 implies $J(\delta^u_{A \sqcap K_2 \sqsubseteq M_2}) \not\models A \sqcap K_2 \sqsubseteq M_2$. Consequently, we have $B \in J(\delta^u_{A \sqcap K_2 \sqsubseteq M_2})$, as required.

- Assume that $\forall \mathsf{inv}(R).C \in J(\delta^u_{A \sqcap K_2 \sqsubseteq M_2})$. Then, we have $\forall \mathsf{inv}(R).C \in \mathbf{L}$ since $J(\delta^u_{A \sqcap K_2 \sqsubseteq M_2})$ is a literal interpretation of $\mathbf{L}$. If $C \notin J(\delta^v_{K_1 \sqsubseteq M_1})$, then $\forall \mathsf{inv}(R).C \in M_2$ by (D.12), so $\forall \mathsf{inv}(R).C \notin J(\delta^u_{A \sqcap K_2 \sqsubseteq M_2})$ by Lemma D.8, which contradicts our assumption. Consequently, we have $C \in J(\delta^v_{K_1 \sqsubseteq M_1})$, as required. □

Claim D.14. *Pre-interpretation I is a pre-model of $\mathcal{O}$.*

Proof. Consider an arbitrary element $\delta^v_{K \sqsubseteq M} \in \Delta$ and an arbitrary normal clause $\alpha \in \mathcal{O}$. By Lemma D.10, set $\mathcal{S}(v)$ is closed under $\prec_v$-hyperresolution with α, so $J(\delta^v_{K \sqsubseteq M}) \models \alpha$ by Lemma D.9. This holds for arbitrary $\delta^v_{K \sqsubseteq M} \in \Delta$, so we have $I \models \alpha$. Finally, the latter holds for arbitrary $\alpha \in \mathcal{O}$, so I is a pre-model of $\mathcal{O}$. □

Claim D.15. *Pre-interpretation I refutes each query $K \sqsubseteq M$ for which there exists a context $v \in \mathcal{V}$ such that v is complete for $K \sqsubseteq M$, $K \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ for each $L \in K$, and $K \sqsubseteq M \mathrel{\hat{\notin}} \mathcal{S}(v)$.*

Proof. Let $K \sqsubseteq M$ be a query satisfying the preconditions of the claim, and let $M' = M \cap \mathbf{L}$. Then $v \nvdash K \sqsubseteq M'$ holds as well, and this clause is over $\mathbf{L}$ by Claim D.12. But then, $J(\delta^v_{K \sqsubseteq M'}) \not\models K \sqsubseteq M'$ by Lemma D.8. Finally, each literal from $M \setminus M'$ is not from $\mathbf{L}$ and so it cannot occur in $J(\delta^v_{K \sqsubseteq M'})$; thus, $J(\delta^v_{K \sqsubseteq M'}) \not\models K \sqsubseteq M$ holds as well, so I refutes $K \sqsubseteq M$, as required. □

D.2 Proof of Theorem 10.4

We proceed similarly as in the proof of Theorem 8.5 in Section D.1.4. Throughout Appendix D.2 we fix a normalized $\mathcal{ALCI}$ ontology $\mathcal{O}$, an admissible decomposition $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \text{core}, \text{knw}, \text{poss}, \prec, \vartheta \rangle$ of $\mathcal{O}$ and an arbitrary finite set of queries, and a clause system $\mathcal{S}$ for $\mathcal{D}$ satisfying the preconditions of Theorem 10.4; furthermore, we fix $\mathbf{L}$ to be the set of all literals that occur in $\mathcal{O}$, $\mathcal{D}$, or $\mathcal{S}$. We prove Theorem 10.4 by constructing a pre-interpretation that satisfies $\mathcal{O}$ but refutes the relevant queries. To simplify the presentation, given an ϵ-component $\mathcal{W}$ of $\mathcal{D}$, a context $v \in \mathcal{W}$, and a clause $K \sqsubseteq M$, we write $v \nvdash K \sqsubseteq M$ if and only if

- $K \subseteq \text{poss}(v)$, $M \cap \text{poss}(\mathcal{W}) \subseteq \text{poss}(v)$, and M is $\prec_v$-minimal (i.e., v is complete for $K \sqsubseteq M$),
- $K \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(v)$ for each literal $L \in K$, and
- $K \sqsubseteq M \mathrel{\hat{\notin}} \mathcal{S}(v)$.

The theorem holds vacuously if no context $v \in \mathcal{V}$ and no query $K \sqsubseteq M$ exist such that $v \nvdash K \sqsubseteq M$. Thus, we assume that $v \nvdash K \sqsubseteq M$ holds for at least one context $v \in \mathcal{V}$ and query $K \sqsubseteq M$; by the definition of clause strengthening, we then also have $v \nvdash K \sqsubseteq M'$ for $M' = M \cap \mathbf{L}$; the latter clause is over $\mathbf{L}$ by Claim D.12. Let $I = \langle \Delta, E, J \rangle$ be defined as follows.

- Set Δ is the smallest set that contains a distinguished element $\delta^v_{K \sqsubseteq M}$ for each context $v \in \mathcal{V}$ and each clause $K \sqsubseteq M$ over $\mathbf{L}$ such that $v \nvdash K \sqsubseteq M$. Set Δ is not empty due to the above assumption.
- The value of function J is defined on each element $\delta^v_{K \sqsubseteq M} \in \Delta$ as follows. Let $\mathcal{W}$ be the ϵ-component of $\mathcal{D}$ that contains v, and let $w_0, w_1, \ldots, w_n$ be an ordering of the elements of

$\mathcal{W}$ obtained by an arbitrary depth-first traversal of $\mathcal{D}_{\mathcal{W}}$ starting from $w_0 = v$. Conjunctions $K^0, K^1, \ldots, K^n$, disjunctions $M^0, M^1, \ldots, M^n$, and literal interpretations $J^0, J^1, \ldots, J^n$ are defined inductively as follows.

- Case $i = 0$. Let $K^0 \coloneqq K$, let $M^0 \coloneqq M$, and let J^0 be an arbitrary literal interpretation of $\mathbf{L}$ w.r.t. $\mathcal{S}(w_0)$ and $\prec_{w_0}$ refuting $K^0 \sqsubseteq M^0$.
- Case $i > 0$. Let w_j be the parent of w_i in the depth-first traversal of $\mathcal{D}_{\mathcal{W}}$; since w_j is considered before w_i in the depth-first traversal, K^j, M^j, and J^j have been defined. Let K^i and M^i be as follows:

$$K^i \coloneqq \bigsqcap \{C \in \mathrm{poss}(w_j) \cap \mathrm{poss}(w_i) \mid C \in J^j\} \tag{D.14}$$

$$M^i \coloneqq \bigsqcup \{C \in \mathrm{poss}(w_j) \cap \mathrm{poss}(w_i) \mid C \notin J^j\} \tag{D.15}$$

  Furthermore, let J^i be an arbitrary literal interpretation of $\mathbf{L}$ w.r.t. $\mathcal{S}(w_i)$ and $\prec_{w_i}$ refuting $K^i \sqsubseteq M^i$.

Based on these definitions for $J^0, J^1, \ldots, J^n$, the value of function J on element $\delta^v_{K \sqsubseteq M}$ is defined by

$$J(\delta^v_{K \sqsubseteq M}) \coloneqq \mathrm{poss}(\mathcal{W}) \cap \left(\bigcup_{i=0}^{n} J^i \right). \tag{D.16}$$

- Set E is the smallest set containing $\langle \delta^v_{K_1 \sqsubseteq M_1}, \delta^u_{A \sqcap K_2 \sqsubseteq M_2}, R \rangle$ for all ϵ-components $\mathcal{W}$ and $\mathcal{U}$ of $\mathcal{D}$, each element $\delta^v_{K_1 \sqsubseteq M_1} \in \Delta$ with $v \in \mathcal{W}$, each literal $\exists R.A \in J(\delta^v_{K_1 \sqsubseteq M_1})$, and each edge $\langle v, u, \exists R.A \rangle \in \mathcal{E}$ with $u \in \mathcal{U}$ such that $u \nvdash A \sqcap K_2 \sqsubseteq M_2$, where K_2 and M_2 are as specified below.

$$K_2 = \bigsqcap \{B \mid \forall R.B \in J(\delta^v_{K_1 \sqsubseteq M_1})\} \tag{D.17}$$

$$M_2 = \bigsqcup \{\forall \mathsf{inv}(R).C \in \mathrm{poss}(\mathcal{U}) \mid C \notin J(\delta^v_{K_1 \sqsubseteq M_1})\} \tag{D.18}$$

In the last item, we have $K_2 \cup M_2 \subseteq \mathbf{L}$ and $A \in \mathbf{L}$; thus, $u \nvdash A \sqcap K_2 \sqsubseteq M_2$ ensures $\delta^u_{A \sqcap K_2 \sqsubseteq M_2} \in \Delta$, and so E is correctly defined. In contrast, the definition of $J(\delta^v_{K \sqsubseteq M})$ relies on the existence of literal

interpretations that refute certain clauses, and it is not obvious that these literal interpretations exist; hence, the following lemma proves that this is the case.

Claim D.16. *The value of $J(\delta^v_{K\sqsubseteq M})$ is correctly defined for each element $\delta^v_{K\sqsubseteq M} \in \Delta$.*

Proof. Consider arbitrary $\delta^v_{K\sqsubseteq M}$, and let $\mathcal{W}$, $w_0, w_1, \ldots, w_n$, $K^0, K^1, \ldots, K^n$, $M^0, M^1, \ldots, M^n$, and $J^0, J^1, \ldots, J^n$ be as specified above. We next inductively prove that $w_i \nvdash K^i \sqsubseteq M^i$ for each $0 \leq i \leq n$; this is sufficient to guarantee existence of the corresponding literal interpretation J^i. For $i = 0$, we have $w_0 \nvdash K^0 \sqsubseteq M^0$ immediately from the facts that $K^0 = K$, $M^0 = M$, and $\delta^v_{K\sqsubseteq M} \in \Delta$. Consider now an arbitrary integer $i > 0$, and let w_j be the parent of w_i in the depth-first traversal of $\mathcal{D}_\mathcal{W}$. Since the ϵ-edges of $\mathcal{D}$ are symmetric by admissibility condition (E1), the presence of the undirected edge $\{w_j, w_i\}$ in $\mathcal{D}_\mathcal{W}$ implies that $\langle w_j, w_i, \epsilon\rangle \in \mathcal{E}$. Now $w_i \nvdash K^i \sqsubseteq M^i$ follows from the following observations.

- $K^i \subseteq \text{poss}(w_i)$ holds trivially by the definition of K^i.
- $M^i \cap \text{poss}(\mathcal{W}) \subseteq \text{poss}(w_i)$ holds trivially since $M^i \subseteq \text{poss}(w_i)$ by the definition of M^i.
- Disjunction M^i is $\prec_{w_i}$-minimal by admissibility condition (P2) since $\langle w_j, w_i, \epsilon\rangle \in \mathcal{E}$ and we have $M^i \subseteq \text{poss}(w_j) \cap \text{poss}(w_i)$.
- Consider an arbitrary literal $L \in K^i$; we show $K^i \sqsubseteq L \,\hat{\in}\, \mathcal{S}(w_i)$. By the definition of K^i, we have $L \in \text{poss}(w_j) \cap \text{poss}(w_i)$; hence, $\langle w_j, w_i, \epsilon\rangle \in \mathcal{E}$ implies $L \sqsubseteq L \,\hat{\in}\, \mathcal{S}(w_i)$ by condition (I4) of Theorem 10.4, which is stronger than $K^i \sqsubseteq L \,\hat{\in}\, \mathcal{S}(w_i)$.
- Assume for contradiction that $K^i \sqsubseteq M^i \,\hat{\in}\, \mathcal{S}(w_i)$. Since $\langle w_j, w_i, \epsilon\rangle \in \mathcal{E}$ and $\mathcal{S}$ is closed under the Epsilon rule, $\mathcal{S}(w_j)$ is closed under $\prec_{w_j}$-hyperresolution with some strengthening of $K^i \sqsubseteq M^i$. By Lemmas D.5 and D.9, we then have $J^j \models K^i \sqsubseteq M^i$; however, by the definition K^i and M^i we clearly have $J^j \not\models K^i \sqsubseteq M^i$. Thus, $K^i \sqsubseteq M^i \,\hat{\notin}\, \mathcal{S}(w_i)$ holds, as required. □

Next, we prove several important properties of J.

Claim D.17. *Let $\delta^v_{K\sqsubseteq M} \in \Delta$ be an arbitrary element, and let $\mathcal{W}$ be the ϵ-component of $\mathcal{D}$ such that $v \in \mathcal{W}$. Then,*

1. $\text{knw}(\mathcal{W}) \subseteq J(\delta^v_{K\sqsubseteq M}) \subseteq \text{poss}(\mathcal{W})$,

2. $J(\delta^v_{K\sqsubseteq M}) \not\models K \sqsubseteq M$, *and*

3. $J(\delta^v_{K\sqsubseteq M}) \models K' \sqsubseteq M'$ *for each clause* $K' \sqsubseteq M'$ *for which there exists a context* $w \in \mathcal{W}$ *such that* $K' \cup M' \subseteq \mathrm{poss}(w)$ *and* $\mathcal{S}(w)$ *is closed under* $\prec_w$*-hyperresolution with* $K' \sqsubseteq M'$.

Proof. Consider an arbitrary element $\delta^v_{K\sqsubseteq M} \in \Delta$; clearly, we have $v \nvdash K \sqsubseteq M$. Furthermore, let $\mathcal{W}$ be the ϵ-component of $\mathcal{D}$ such that $v \in \mathcal{W}$, and let $w_0, w_1, \ldots, w_n$, $K^0, K^1, \ldots, K^n$, $M^0, M^1, \ldots, M^n$, and $J^0, J^1, \ldots, J^n$ be as specified in the definition of the pre-interpretation I. Before proceeding, we first prove the following auxiliary property $(*)$.

> For all integers $0 \leq i, j \leq n$ and each literal $L \in \mathrm{poss}(w_i) \cap \mathrm{poss}(w_j)$, we have $L \in J^i$ if and only if $L \in J^j$.

Since $\mathcal{D}_{\mathcal{W}}$ is a tree by admissibility condition (E2), the shortest path between w_i and w_j is unique; we now prove $(*)$ by induction on the length of this path. Towards this goal, consider an arbitrary literal $L \in \mathrm{poss}(w_i) \cap \mathrm{poss}(w_j)$; we have the following cases.

- If $i = j$, then $J^i = J^j$, so clearly $L \in J^i$ if and only if $L \in J^j$.
- Assume that w_j is the parent of w_i in the traversal of $\mathcal{D}_{\mathcal{W}}$. Then $K^i \cup M^i = \mathrm{poss}(w_i) \cap \mathrm{poss}(w_j)$ by the definition of K^i and M^i, which implies $L \in K^i \cup M^i$. Furthermore, by Lemma D.8, we have $J^i \not\models K^i \sqsubseteq M^i$. But then, if $L \in K^i$, then by the definition of K^i we have $L \in J^j$, and $J^i \not\models K^i \sqsubseteq M^i$ implies $L \in J^i$. Analogously, if $L \in M^i$, then by the definition of M^i we have $L \notin J^j$, and $J^i \not\models K^i \sqsubseteq M^i$ implies $L \notin J^i$.
- The case when w_i is the parent of w_j is symmetric to the previous one.
- The remaining possibility is that the length of the shortest path between w_i and w_j is greater than one, so let w_k be an arbitrary vertex on this path different from w_i and w_j. By admissibility condition (E3), we have $L \in \mathrm{poss}(w_k)$. Furthermore, the paths between w_i and w_k, and between w_k and w_j are both shorter than the path between w_i and w_j, so property $(*)$ holds for i and k, and for k and j. But then, property $(*)$ clearly holds for i and j as well.

Property $(*)$ clearly implies that, for each integer $0 \leq i \leq n$ and each literal $L \in \mathrm{poss}(w_i)$, we have $L \in J(\delta^v_{K\sqsubseteq M})$ if and only if $L \in J^i$; we call this property $(\dagger)$. We are now ready to prove Properties 1–3 of this claim.

(Property 1) The definition of $J(\delta^v_{K\sqsubseteq M})$ in (D.16) clearly implies $J(\delta^v_{K\sqsubseteq M}) \subseteq \text{poss}(\mathcal{W})$. In order to prove $\text{knw}(\mathcal{W}) \subseteq J(\delta^v_{K\sqsubseteq M})$, consider an arbitrary literal $L \in \text{knw}(\mathcal{W})$; clearly, a context $w_i \in \mathcal{W}$ exists such that $L \in \text{knw}(w_i)$. By condition (I2) of Theorem 10.4, we have $\top \sqsubseteq L \,\hat{\in}\, \mathcal{S}(w_i)$; hence $L \in J^i$ by Lemma D.7. Consequently, $L \in J(\delta^v_{K\sqsubseteq M})$ holds by property (†), as required.

(Property 2) By Lemma D.8, $K^0 = K$, and $M^0 = M$, we have $J^0 \not\models K \sqsubseteq M$—that is, $K \subseteq J^0$ and $M \cap J^0 = \emptyset$. Now $v \nvdash K \sqsubseteq M$ implies $K \subseteq \text{poss}(v)$ and $M \cap \text{poss}(\mathcal{W}) \subseteq \text{poss}(v)$, so property (†) implies $K \subseteq J(\delta^v_{K\sqsubseteq M})$ and $[M \cap \text{poss}(\mathcal{W})] \cap J(\delta^v_{K\sqsubseteq M}) = \emptyset$; since $\text{poss}(\mathcal{W}) \cap J(\delta^v_{K\sqsubseteq M}) = J(\delta^v_{K\sqsubseteq M})$, we have $M \cap J(\delta^v_{K\sqsubseteq M}) = \emptyset$. Hence $J(\delta^v_{K\sqsubseteq M}) \not\models K \sqsubseteq M$ holds, as required.

(Property 3) Let $K' \sqsubseteq M'$ be a clause and let $w_i \in \mathcal{W}$ be a context such that $K' \cup M' \subseteq \text{poss}(w_i)$ and set $\mathcal{S}(w_i)$ is closed under $\prec_{w_i}$-hyperresolution with $K' \sqsubseteq M'$. By Lemma D.9, $J^i \models K' \sqsubseteq M'$. But then, since $K' \cup M' \subseteq \text{poss}(w_i)$, property (†) implies $J(\delta^v_{K\sqsubseteq M}) \models K' \sqsubseteq M'$, as required. □

To complete the proof of Theorem 10.4, the following claims show that structure I is a pre-interpretation, and that it satisfies $\mathcal{O}$ but refutes the relevant queries.

Claim D.18. *Structure I satisfies conditions* ($I_\exists$) *and* ($I_\forall$) *of pre-interpretations from Definition D.1.*

Proof. (Condition $I_\exists$) Consider an arbitrary element $\delta^v_{K_1\sqsubseteq M_1} \in \Delta$ and a literal $\exists R.A \in J(\delta^v_{K_1\sqsubseteq M_1})$; we show that an edge $\langle \delta^v_{K_1\sqsubseteq M_1}, \gamma, R\rangle \in E$ exists such that $A \in J(\gamma)$. Let $\mathcal{W}$ be the ϵ-component of $\mathcal{D}$ such that $v \in \mathcal{W}$. By admissibility condition (S3), an edge $\langle w, u, \exists R.A\rangle \in \mathcal{E}$ exists such that $w \in \mathcal{W}$ and conditions (D.19) and (D.20) are both satisfied.

$$\text{poss}(u) \supseteq \{A\} \cup \{B \mid \forall R.B \in \text{poss}(\mathcal{W})\} \tag{D.19}$$

$$\text{poss}(w) \supseteq \{\exists R.A\} \cup \{\forall R.B \mid \forall R.B \in \text{poss}(\mathcal{W})\} \tag{D.20}$$

Let $\mathcal{U}$ be the ϵ-component of $\mathcal{D}$ such that $u \in \mathcal{U}$, and let K_2 and M_2 be as in (D.17) and (D.18). The following observations establish $u \nvdash A \sqcap K_2 \sqsubseteq M_2$.

- $A \sqcap K_2 \subseteq \text{poss}(u)$ follows from (D.17), Property 1 of Claim D.17, and (D.19).
- Admissibility condition (S2) implies $M_2 \cap \text{poss}(\mathcal{U}) \subseteq \text{poss}(u)$.
- Disjunction M_2 is $\prec_u$-minimal since, by admissibility condition (P1), $\langle w, u, \exists R.A\rangle \in \mathcal{E}$ implies that $\prec_u$ is R-admissible.

- Consider an arbitrary literal $L \in A \sqcap K_2$; we show that $A \sqcap K_2 \sqsubseteq L \mathrel{\hat{\in}} \mathcal{S}(u)$. Assume that $L \in K_2$, so L is an atomic concept $L = B$. Then, we have $\forall R.B \in J(\delta^v_{K_1 \sqsubseteq M_1}) \subseteq \mathrm{poss}(\mathcal{W})$ by (D.17) and Property 1 of Claim D.17. Thus, we have $B \in \mathrm{poss}(u)$ and $\forall R.B \in \mathrm{poss}(w)$ by (D.19) and (D.20), and then $B \sqsubseteq B \mathrel{\hat{\in}} \mathcal{S}(u)$ by condition (I3) of Theorem 10.4, which is stronger than the required $A \sqcap K_2 \sqsubseteq B \mathrel{\hat{\in}} \mathcal{S}(u)$. The proof for the case $L = A$ is analogous.

- Assume for contradiction that $A \sqcap K_2 \sqsubseteq M_2 \mathrel{\hat{\in}} \mathcal{S}(u)$. By admissibility condition (S2), we have $C \in \mathrm{poss}(w)$ for each $\forall \mathsf{inv}(R).C \in M_2$. Thus, since the Pred rule from Table 10.1 is not applicable to $\mathcal{S}(v)$, the Pred from Table 8.1 is not applicable to $\mathcal{S}(v)$ either. Hence, by Lemma D.11, set $\mathcal{S}(w)$ is closed under $\prec_w$-hyperresolution with the clause

$$\exists R.A \sqcap \bigsqcap \{\forall R.B \mid \forall R.B \in J(\delta^v_{K_1 \sqsubseteq M_1})\} \sqsubseteq \bigsqcup \{C \mid \forall \mathsf{inv}(R).C \in \mathrm{poss}(\mathcal{U}) \text{ and } C \notin J(\delta^v_{K_1 \sqsubseteq M_1})\}.$$

  Recall that $\exists R.A \in J(\delta^v_{K_1 \sqsubseteq M_1})$, so the above clause is clearly refuted in $J(\delta^v_{K_1 \sqsubseteq M_1})$. However, all literals on the left-hand side of this clause are contained in $\mathrm{poss}(w)$ by (D.20), and each concept C on its right-hand side is contained in $\mathrm{poss}(w)$; but then, Property 3 of Claim D.17 implies that this clause is satisfied in $J(\delta^v_{K_1 \sqsubseteq M_1})$, which is a contradiction. Consequently, we have $A \sqcap K_2 \sqsubseteq M_2 \mathrel{\hat{\notin}} \mathcal{S}(u)$, as required.

Thus, we have $u \nvdash A \sqcap K_2 \sqsubseteq M_2$, which implies $\langle \delta^v_{K_1 \sqsubseteq M_1}, \delta^u_{A \sqcap K_2 \sqsubseteq M_2}, R \rangle \in E$ by the definition of E. Finally, $A \in J(\delta^u_{A \sqcap K_2 \sqsubseteq M_2})$ holds by Property 2 of Claim D.17. This concludes the proof of condition $(I_\exists)$.

(Condition $I_\forall$). Consider an arbitrary edge $\langle \delta^v_{K_1 \sqsubseteq M_1}, \delta^u_{A \sqcap K_2 \sqsubseteq M_2}, R \rangle \in E$ and arbitrary literals $\forall R.B$ and $\forall \mathsf{inv}(R).C$, and let $\mathcal{U}$ be the ϵ-component of $\mathcal{D}$ such that $u \in \mathcal{U}$.

- Assume that $\forall R.B \in J(\delta^v_{K_1 \sqsubseteq M_1})$. Then $B \in K_2$ by (D.17); hence Property 2 of Claim D.17 implies $J(\delta^u_{A \sqcap K_2 \sqsubseteq M_2}) \not\models A \sqcap K_2 \sqsubseteq M_2$; therefore $B \in J(\delta^u_{A \sqcap K_2 \sqsubseteq M_2})$, as required.

- Assume that $\forall \mathsf{inv}(R).C \in J(\delta^u_{A \sqcap K_2 \sqsubseteq M_2})$. Then, we have $\forall \mathsf{inv}(R).C \in \mathrm{poss}(\mathcal{U})$ by Property 1 of Claim D.17. If $C \notin J(\delta^v_{K_1 \sqsubseteq M_1})$, then $\forall \mathsf{inv}(R).C \in M_2$ by (D.18), so $\forall \mathsf{inv}(R).C \notin J(\delta^u_{A \sqcap K_2 \sqsubseteq M_2})$ by Property 2 of Claim D.17, which contradicts the assumption. Consequently, we have $C \in J(\delta^v_{K_1 \sqsubseteq M_1})$, as required. □

Claim D.19. *Pre-interpretation I is a pre-model of $\mathcal{O}$.*

Proof. Consider an arbitrary element $\delta^v_{K\sqsubseteq M} \in \Delta$ and an arbitrary normal clause $K' \sqsubseteq M' \in \mathcal{O}$; we show that $J(\delta^v_{K\sqsubseteq M})$ satisfies $K' \sqsubseteq M'$. This is obvious if $K' \not\subseteq J(\delta^v_{K\sqsubseteq M})$, so assume $K' \subseteq J(\delta^v_{K\sqsubseteq M})$. Let $\mathcal{W}$ be the ϵ-component of $\mathcal{D}$ such that $v \in \mathcal{W}$. Then, by Property 1 of Claim D.17, we have $K' \subseteq \text{poss}(\mathcal{W})$. Furthermore, by the ontology condition of Definition 10.3, a context $w \in \mathcal{W}$ exists such that $K' \cup M' \subseteq \text{poss}(w)$. Since $\mathcal{S}$ is closed under the Hyper rule from Table 10.1, set $\mathcal{S}(w)$ is closed under $\prec_w$-hyperresolution with $K' \sqsubseteq M'$. But then, Property 3 of Claim D.17 implies $J(\delta^v_{K\sqsubseteq M}) \models K' \sqsubseteq M'$, as required. □

Claim D.20. *Pre-interpretation I refutes each query $K \sqsubseteq M$ for which there exists a context $v \in \mathcal{V}$ such that v is complete for $K \sqsubseteq M$, $K \sqsubseteq L \,\hat{\in}\, \mathcal{S}(v)$ for each $L \in K$, and $K \sqsubseteq M \,\hat{\notin}\, \mathcal{S}(v)$.*

Proof. Let $K \sqsubseteq M$ be a query satisfying the preconditions of the claim, and let $M' = M \cap \mathbf{L}$. Then $v \nvdash K \sqsubseteq M'$ holds as well, and this clause is over $\mathbf{L}$ by Claim D.12. But then, $J(\delta^v_{K\sqsubseteq M'}) \not\models K \sqsubseteq M'$ by Property 2 of Claim D.17. Finally, each literal from $M \setminus M'$ is not from $\mathbf{L}$ so it cannot occur in $J(\delta^v_{K\sqsubseteq M'})$; thus, $J(\delta^v_{K\sqsubseteq M'}) \not\models K \sqsubseteq M$ holds as well, so I refutes $K \sqsubseteq M$, as required. □

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1994.

[2] Eyal Amir and Sheila A. McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence*, 162(1-2):49–88, 2005.

[3] Hajnal Andréka, István Németi, and Johan van Benthem. Modal languages and bounded fragments of predicate logic. *J. of Philosophical Logic*, 27(3):217–274, 1998.

[4] Ana Armas Romero, Bernardo Cuenca Grau, and Ian Horrocks. MORe: modular combination of OWL reasoners for ontology classification. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *Proc. 11th Int. Semantic Web Conf. (ISWC'12)*, volume 7649 of *LNCS*, pages 1–16. Springer, 2012.

[5] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The DL-Lite family and relations. *J. of Artificial Intelligence Research*, 36:1–69, 2009.

[6] Mina Aslani and Volker Haarslev. Parallel TBox classification in description logics – first experimental results. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proc. 19th European Conf. on Artificial Intelligence (ECAI'10)*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 485–490. IOS Press, 2010.

[7] Mina Aslani and Volker Haarslev. Concurrent classification of OWL ontologies – An empirical evaluation. In Yevgeny Kazakov, Domenico Lembo, and Frank Wolter, editors, *Proc. 25th Int. Workshop on Description Logics (DL'12)*, volume 846 of *CEUR Workshop Proceedings*, pages 400–410. CEUR-WS.org, 2012.

[8] Franz Baader. Terminological cycles in a description logic with existential restrictions. In Georg Gottlob and Toby Walsh, editors, *Proc. 18th Int. Joint Conf. on Artificial Intelligence (IJCAI'03)*, pages 325–330. Morgan Kaufmann, 2003.

[9] Franz Baader, Stefan Borgwardt, and Barbara Morawska. Extending unification in $\mathcal{EL}$ towards general TBoxes. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Proc. 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'12)*, pages 568–572. AAAI Press, 2012.

[10] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05)*, pages 364–369. Professional Book Center, 2005.

[11] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope further. In Kendall G. Clark and Peter F. Patel-Schneider, editors, *Proc. OWLED 2008 DC Workshop on OWL: Experiences and Directions*, volume 496 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[12] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, second edition, 2007.

[13] Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In Thomas Dean, editor, *Proc. 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 96–103. Morgan Kaufmann, 1999.

[14] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. 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.

[15] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In Ulrich Furbach and Natarajan Shankar, editors, *Proc. 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06)*, volume 4130 of *LNCS*, pages 287–291. Springer, 2006.

[16] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. Efficient reasoning in $\mathcal{EL}^+$. In Bijan Parsia, Ulrike Sattler, and David Toman, editors, *Proc. 19th Int. Workshop on Description Logics (DL'06)*, volume 189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[17] Franz Baader, Baris Sertkaya, and Anni-Yasmin Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. of Applied Logics*, 5(3):392–420, 2007.

[18] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, volume I, chapter 2, pages 19–99. Elsevier and MIT Press, 2001.

[19] Matteo Baldoni, Laura Giordano, and Alberto Martelli. A tableau calculus for multimodal logics and some (un)decidability results. In *Proc. Int. Conf. on Automatic Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'98)*, pages 44–59. Springer, 1998.

[20] Dave Beckett, editor. *RDF/XML Syntax Specification (Revised)*. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-syntax-grammar/`.

[21] Frank W. Bergmann and Joachim Quantz. Parallelizing description logics. In Ipke Wachsmuth, Claus-Rainer Rollinger, and Wilfried Brauer, editors, *Proc. 19th Annual GermanConf. on Artificial Intelligence (KI'95)*, volume 981 of *LNCS*, pages 137–148. Springer, 1995.

[22] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[23] Loris Bozzato, Martin Homola, and Luciano Serafini. Towards more effective tableaux reasoning for CKR. In Yevgeny Kazakov, Domenico Lembo, and Frank Wolter, editors, *Proc. 25th Int. Workshop on Description Logics (DL'12)*, volume 846 of *CEUR Workshop Proceedings*, pages 114–124. CEUR-WS.org, 2012.

[24] Ronald J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In Ronald J. Brachman, editor, *Proc. 4th National Conf. on Artificial Intelligence (AAAI'84)*, pages 34–37. AAAI Press, 1984.

[25] Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and – What else? In Ramon López de Mántaras and Lorenza Saitta, editors, *Proc. 16th European Conf. on Artificial Intelligence (ECAI'04)*, pages 298–302. IOS Press, 2004.

[26] Dan Brickley and Ramanathan V. Guha, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-schema/`.

[27] Diego Calvanese, Guiseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[28] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular reuse of ontologies: Theory and practice. *J. of Artificial Intelligence Research*, 31:273–318, 2008.

[29] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *J. of Web Semantics*, 6:309–322, 2008.

[30] John Day-Richter, editor. *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`.

[31] Vincent Delaitre and Yevgeny Kazakov. Classifying $\mathcal{ELH}$ ontologies in SQL databases. In Peter F. Patel-Schneider and Rinke Hoekstra, editors, *Proc. OWLED 2009 Workshop on OWL: Experiences and Directions*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[32] Stéphane Demri and Hans Nivelle. Deciding regular grammar logics with converse through first-order logic. *J. of Logic, Language and Information*, 14(3):289–329, 2005.

[33] Sebastian Derriere, Alexandre Richard, and Andrea Preite-Martinez. An ontology of astronomical object types for the virtual observatory. In *Proc. 26th meeting of the IAU: Virtual Observatory in Action: New Science, New Technology, and Next Generation Facilities*, pages 17–18, 2006.

[34] Rod G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.

[35] Thomas Eiter, Thomas Krennwallner, Patrik Schneider, and Guohui Xiao. Uniform evaluation of nonmonotonic DL-programs. In Thomas Lukasiewicz and Attila Sali, editors, *Proc. 7th Int. Symposium on Foundations of Information and Knowledge Systems (FoIKS'12)*, volume 7153 of *LNCS*, pages 1–22. Springer, 2012.

[36] Christian G. Fermüller, Alexander Leitsch, Ullrich Hustadt, and Tanel Tammet. Resolution decision procedures. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, volume II, chapter 25, pages 1791–1849. Elsevier and MIT Press, 2001.

[37] Charles Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.

[38] Georgios V. Gkoutos, Paul N. Schofield, and Robert Hoehndorf. Computational tools for comparative phenomics: the role and promise of ontologies. *Mammalian Genome*, 23(9–10):669–679, 2012.

[39] Birte Glimm, Ian Horrocks, Boris Motik, Rob Shearer, and Giorgos Stoilos. A novel approach to ontology classification. *J. of Web Sementics*, 14:84–101, 2012.

[40] Christine Golbreich, Songmao Zhang, and Oliver Bodenreider. The foundational model of anatomy in OWL: Experience and perspectives. *J. of Web Semantics*, 4(3):181–195, 2006.

[41] Rafael S. Gonçalves, Bijan Parsia, and Ulrike Sattler. Performance heterogeneity and approximate reasoning in description logic ontologies. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *Proc. 11th Int. Semantic Web Conf. (ISWC'12)*, volume 7649 of *LNCS*, pages 82–98. Springer, 2012.

[42] John Goodwin. Experiences of using OWL at the Ordnance Survey. In Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Peter F. Patel-Schneider, editors, *Proc. OWLED 2005 Workshop on OWL: Experiences and Directions*, volume 188 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[43] Rajeev Goré and Linh Anh Nguyen. EXPTIME tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In Nicola Olivetti, editor, *Proc. 16th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 4548 of *LNCS*, pages 133–148. Springer, 2007.

[44] Rajeev Goré and Linh Anh Nguyen. ExpTime tableaux for $\mathcal{ALC}$ using sound global caching. *Journal of Automated Reasoning*, 50(4):355–381, 2013.

[45] Stephan Grimm, Michael Watzke, Thomas Hubauer, and Falco Cescolini. Embedded $\mathcal{EL}^+$ reasoning on programmable logic controllers. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *Proc. 11th Int. Semantic Web Conf. (ISWC'12)*, volume 7649 of *LNCS*, pages 66–81. Springer, 2012.

[46] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *Proc. 12th Int. Conf. on World Wide Web (WWW'03)*, pages 48–57. ACM, 2003.

[47] Volker Haarslev and Ralf Möller. Racer system description. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proc. 1st Int. Joint Conf. on Automated Reasoning (IJCAR'01)*, volume 2083 of *LNCS*, pages 701–705. Springer, 2001.

[48] Patrick Hayes, editor. *RDF Semantics*. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-mt/`.

[49] Norman Heino and Jeff Z. Pan. RDFS reasoning on massively parallel hardware. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein,

and Eva Blomqvist, editors, *Proc. 11th Int. Semantic Web Conf. (ISWC'12)*, volume 7649 of *LNCS*, pages 133–148. Springer, 2012.

[50] Robert Hoehndorf, Michel Dumontier, and Georgios V. Gkoutos. Identifying aberrant pathways through integrated analysis of knowledge in pharmacogenomics. *Bioinformatics*, 28(16):2169–2175, 2012.

[51] Robert Hoehndorf, Midori A. Harris, Heinrich Herre, Gabriella Rustici, and Georgios V. Gkoutos. Semantic integration of physiology phenotypes with an application to the cellular phenotype ontology. *Bioinformatics*, 28(13):1783–1789, 2012.

[52] Martin Hofmann. Proof-theoretic approach to description-logic. In *Proc. 20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*, pages 229–237. IEEE Computer Society, 2005.

[53] Aidan Hogan, Andreas Harth, and Axel Polleres. Scalable authoritative OWL reasoning for the Web. *Int. J. of Semantic Web Inf. Syst.*, 5(2):49–90, 2009.

[54] Aidan Hogan, Jeff Z. Pan, Axel Polleres, and Stefan Decker. SAOR: template rule optimisations for distributed reasoning over 1 billion linked data triples. In Peter F. Patel-Schneider, Yue Pan, Birte Glimm, Pascal Hitzler, Peter Mika, Jeff Pan, and Ian Horrocks, editors, *Proc. 9th Int. Semantic Web Conf. (ISWC'10)*, volume 6496 of *LNCS*, pages 337–353. Springer, 2010.

[55] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for working with OWL 2 ontologies. In Peter F. Patel-Schneider and Rinke Hoekstra, editors, *Proc. OWLED 2009 Workshop on OWL: Experiences and Directions*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[56] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 57–67. AAAI Press, 2006.

[57] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David A. McAllester, and Andrei Voronkov, editors, *Proc. 6th Int. Conf. on Logic Programming and Automated Reasoning (LPAR'99)*, volume 1705 of *LNCS*, pages 161–180. Springer, 1999.

[58] Simon Jupp, Robert Stevens, and Robert Hoehndorf. Logical gene ontology annotations (GOAL): exploring gene ontology annotations with OWL. *J. of Biomedical Semantics*, 3(Suppl 1)(S3):1–16, 2012.

[59] Yevgeny Kazakov. $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In Gerhard Brewka and Jérôme Lang, editors, *Proc. 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 274–284. AAAI Press, 2008.

[60] Yevgeny Kazakov. Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In Craig Boutilier, editor, *Proc. 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09)*, pages 2040–2045. IJCAI, 2009.

[61] Yevgeny Kazakov. An extension of regularity conditions for complex role inclusion axioms. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Proc. 22nd Int. Workshop on Description Logics (DL'09)*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[62] Yevgeny Kazakov and Pavel Klinov. Incremental reasoning in OWL EL without bookkeeping. Technical report, University of Ulm, 2013. Available at `http://code.google.com/p/elk-reasoner/`.

[63] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. Concurrent classification of $\mathcal{EL}$ ontologies. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist, editors, *Proc. 10th Int. Semantic Web Conf. (ISWC'11)*, volume 7032 of *LNCS*, pages 305–320. Springer, 2011.

[64] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. Unchain my $\mathcal{EL}$ reasoner. In Riccardo Rosati, Sebastian Rudolph, and Michael Zakharyaschev, editors, *Proc. 24th Int. Workshop on Description Logics (DL'11)*, volume 745 of *CEUR Workshop Proceedings*, pages 202–212. CEUR-WS.org, 2011.

[65] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. Practical reasoning with nominals in the $\mathcal{EL}$ family of description logics. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Proc. 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'12)*, pages 264–274. AAAI Press, 2012.

[66] Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, and Mark A. Musen. The Protégé OWL Plugin: An open development environment for Semantic Web applications. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proc. 3rd Int. Semantic Web Conf. (ISWC'04)*, volume 3298 of *LNCS*, pages 229–243. Springer, 2004.

[67] Vladimir Kolovski, Zhe Wu, and George Eadon. Optimizing enterprise-scale OWL 2 RL reasoning in a relational database system. In Peter F. Patel-Schneider, Yue Pan, Birte Glimm, Pascal Hitzler, Peter Mika, Jeff Pan, and Ian Horrocks, editors, *Proc. 9th Int. Semantic Web Conf. (ISWC'10)*, volume 6496 of *LNCS*, pages 436–452. Springer, 2010.

[68] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to ontology-based data access. In Toby Walsh, editor, *Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11)*, pages 2656–2661. AAAI Press/IJCAI, 2011.

[69] Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Mind the data skew: distributed inferencing by speeddating in elastic regions. In *Proc. 19th Int. Conf. on World Wide Web (WWW'10)*, WWW'10, pages 531–540. ACM, 2010.

[70] Markus Krötzsch. Efficient rule-based inferencing for OWL EL. In Toby Walsh, editor, *Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11)*, pages 2668–2673. AAAI Press/IJCAI, 2011.

[71] Markus Krötzsch. The not-so-easy task of computing class subsumptions in OWL RL. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *Proc. 11th Int. Semantic Web Conf. (ISWC'12)*, volume 7649 of *LNCS*, pages 279–294. Springer, 2012.

[72] Markus Krötzsch. OWL 2 Profiles: An introduction to lightweight ontology languages. In Thomas Eiter and Thomas Krennwallner, editors, *Proc. 8th Reasoning Web Summer School, Vienna, Austria, September 3–8 2012*, volume 7487 of *LNCS*, pages 112–183. Springer, 2012.

[73] Markus Krötzsch, Anees Mehdi, and Sebastian Rudolph. Orel: Database-driven reasoning for OWL 2 profiles. In Volker Haarslev, David Toman, and Grant Weddell, editors, *Proc. 23rd Int. Workshop on Description Logics (DL'10)*, volume 573 of *CEUR Workshop Proceedings*, pages 114–124. CEUR-WS.org, 2010.

[74] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Complexity boundaries for Horn description logics. In *Proc. 22nd AAAI Conf. on Artificial Intelligence (AAAI'07)*, pages 452–457. AAAI Press, 2007.

[75] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Conjunctive queries for a tractable fragment of OWL 1.1. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *Proc. 6th Int. Semantic Web Conf. (ISWC'07)*, volume 4825 of *LNCS*, pages 310–323. Springer, 2007.

[76] Markus Krötzsch, František Simančík, and Ian Horrocks. A description logic primer. *CoRR*, abs/1201.4089, 2012.

[77] Lee Lacy, Gabriel Aviles, Karen Fraser, William Gerber, Alice M. Mulvehill, and Robert Gaskill. Experiences using OWL in military applications. In Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Peter F. Patel-Schneider, editors, *Proc. OWLED 2005 Workshop on OWL: Experiences and Directions*, volume 188 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[78] Michael J. Lawley and Cyril Bousquet. Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In Kerry Taylor, Thomas Meyer, and Mehmet Orgun, editors, *Proc. 6th Australasian Ontology Workshop (IAOA'10)*, volume 122 of *Conferences in Research and Practice in Information Technology*, pages 45–49. Australian Computer Society Inc., 2010.

[79] Fritz Lehmann. Semantic networks. In Fritz Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 1–50. Pergamon Press, 1992.

[80] Thorsten Liebig and Felix Müller. Parallelizing tableaux-based description logic reasoning. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *Proceedings of OTM Workshops 2007, Part II*, volume 4806 of *LNCS*, pages 1135–1144. Springer, 2007.

[81] Ewing L. Lusk, William McCune, and John K. Slaney. Roo: A parallel theorem prover. In Deepak Kapur, editor, *Proc. 11th Conf. on Automated Deduction (CADE'92)*, volume 607 of *LNCS*, pages 731–734. Springer, 1992.

[82] Carsten Lutz, Dirk Walther, and Frank Wolter. Conservative extensions in expressive description logics. In Manuela M. Veloso, editor, *Proc. 20th Int. Joint Conf. on Artificial Intelligence (IJCAI'07)*, pages 453–458. IJCAI, 2007.

[83] Despoina Magka, Yevgeny Kazakov, and Ian Horrocks. Tractable extensions of the description logic $\mathcal{EL}$ with numerical datatypes. *J. of Automated Reasoning*, 47(4):427–450, 2011.

[84] Frank Manola and Eric Miller, editors. *Resource Description Framework (RDF): Primer*. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-primer/`.

[85] Adam Meissner. 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.

[86] Julian Mendez. jcel: A modular rule-based reasoner. In Ian Horrocks, Mikalai Yatskevich, and Ernesto Jimenez-Ruiz, editors, *Proc. OWL Reasoner Evaluation Workshop 2012 (ORE'12)*, volume 858 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[87] Julian Mendez, Andreas Ecke, and Anni-Yasmin Turhan. Implementing completion-based inferences for the $\mathcal{EL}$-family. In Riccardo Rosati, Sebastian Rudolph, and Michael Zakharyaschev, editors, *Proc. 24th Int. Workshop on Description Logics (DL'11)*, volume 745 of *CEUR Workshop Proceedings*, pages 334–344. CEUR-WS.org, 2011.

[88] Julian Mendez and Boontawee Suntisrivaraporn. Reintroducing CEL as an OWL 2 EL reasoner. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Proc. 22nd Int. Workshop on Description Logics (DL'09)*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[89] Ralf Möller, Volker Haarslev, and Sebastian Wandelt. The revival of structural subsumption in tableau-based reasoners. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *Proc. 21st Int. Workshop on Description Logics (DL'08)*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[90] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz, editors. *OWL 2 Web Ontology Language: Profiles*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-profiles/`.

[91] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau, editors. *OWL 2 Web Ontology Language: Direct Semantics*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-direct-semantics/`.

[92] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia, editors. *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-syntax/`.

[93] Boris Motik and Ulrike Sattler. A comparison of reasoning techniques for querying large description logic ABoxes. In Miki Hermann and Andrei Voronkov, editors, *Proc. 13th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'06)*, volume 4246 of *LNCS*, pages 227–241. Springer, 2006.

[94] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau reasoning for description logics. *J. of Artificial Intelligence Research*, 36:165–228, 2009.

[95] Raghava Mutharaju, Frederick Maier, and Pascal Hitzler. A MapReduce algorithm for $\mathcal{EL}^+$. In Volker Haarslev, David Toman, and Grant Weddell, editors, *Proc. 23rd Int. Workshop on Description Logics (DL'10)*, volume 573 of *CEUR Workshop Proceedings*, pages 464–474. CEUR-WS.org, 2010.

[96] Sivaramakrishnan Narayanan, Ümit V. Çatalyürek, Tahsin M. Kurç, and Joel H. Saltz. Parallel materialization of large ABoxes. In Sung Y. Shin and Sascha Ossowski, editors, *Proc. ACM Symposium on Applied Computing (SAC'09)*, pages 1257–1261. ACM, 2009.

[97] Nadeschda Nikitina and Sebastian Rudolph. ExpExpExplosion: uniform interpolation in general $\mathcal{EL}$ terminologies. In Luc De Raedt, Christian Bessière, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, editors, *Proc. 20th European Conf. on Artificial Intelligence (ECAI'12)*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 618–623. IOS Press, 2012.

[98] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski, editors, *Proc. 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'10)*, pages 269–279. AAAI Press, 2010.

[99] David Osumi-Sutherland, Simon Reeve, Christopher J. Mungall, Fabian Neuhaus, Alan Ruttenberg, Gregory S. X. E. Jefferis, and J. Douglas Armstrong. A strategy for building neuroanatomy ontologies. *Bioinformatics*, 28(9):1262–1269, 2012.

[100] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-overview/`.

[101] Antonella Poggi, Diego Calvanese, De Giacomo, Giuseppe, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. of Data Semantics*, X:133–173, 2008.

[102] Alan Rector, Also Gangemi, Elena Galeazzi, Andrzej J. Glowinski, and Aangelo Rossi-Mori. The GALEN CORE model schemata for anatomy: Towards a re-usable application-independent model of medical concepts. In Pedro Barahona, Mario Veloso, and Jeremy Bryant, editors, *Proc. 12th Int. Congress of the European Federation for Medical Informatics (MIE'94)*, pages 229–233, 1994.

[103] Alan Rector and Luigi Iannone. 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.

[104] Alan L. Rector, Sean Bechhofer, Carole A. Goble, Ian Horrocks, W. A. Nowlan, and W. D. Solomon. The grail concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9(2):139–171, 1997.

[105] Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *J. of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

[106] Jeremy E. Rogers. Quality assurance of medical ontologies. *Methods Inf Med.*, 45(3):267–274, 2006.

[107] Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. Cheap Boolean role constructors for description logics. In Steffen Hölldobler, Carsten Lutz, and Heinrich Wansing, editors, *Proc. 11th European Conf. on Logics in Artificial Intelligence (JELIA'08)*, volume 5293 of *LNAI*, pages 362–374. Springer, 2008.

[108] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In John Mylopoulos and Raymond Reiter, editors, *Proc. 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471. Morgan Kaufmann, 1991.

[109] Anne Schlicht and Heiner Stuckenschmidt. Peer-to-peer reasoning for interlinked ontologies. *Int. J. of Semantic Computing*, 4(1):27–58, 2010.

[110] Anne Schlicht and Heiner Stuckenschmidt. MapResolve. In Sebastian Rudolph and Claudio Gutierrez, editors, *Proc. 5th Int. Conf. on Web Reasoning and Rule Systems (RR'11)*, volume 6902 of *LNCS*, pages 294–299. Springer, 2011.

[111] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *J. of Artificial Intelligence*, 48:1–26, 1991.

[112] Michael Schneider, editor. *OWL 2 Web Ontology Language: RDF-Based Semantics*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-rdf-based-semantics/`.

[113] Stefan Schulz, Ronald Cornet, and Kent A. Spackman. Consolidating SNOMED CT's ontological commitment. *Applied Ontology*, 6(1):1–11, 2011.

[114] Stepahn Schulz. System description: E 0.81. In David A. Basin and Michael Rusinowitch, editors, *Proc. 2nd Int. Joint Conf. on Automated Reasoning (IJCAR'04)*, volume 3097 of *LNCS*, pages 223–228. Springer, 2004.

[115] Christian Seitz and René Schönfelder. Rule-based OWL reasoning for specific embedded devices. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist, editors, *Proc. 10th Int. Semantic Web Conf. (ISWC'11)*, volume 7032 of *LNCS*, pages 237–252. Springer, 2011.

[116] Baris Sertkaya. In the search of improvements to the $\mathcal{EL}^+$ classification algorithm. In Riccardo Rosati, Sebastian Rudolph, and Michael Zakharyaschev, editors, *Proc. 24th Int. Workshop on Description Logics (DL'11)*, volume 745 of *CEUR Workshop Proceedings*, pages 389–399. CEUR-WS.org, 2011.

[117] Eep S. Sidhu, Tharam S. Dillon, Elizabeth Chang, and Baldev S. Sidhu. Protein ontology development using OWL. In Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Peter F. Patel-Schneider, editors, *Proc. OWLED 2005 Workshop on OWL: Experiences and Directions*, volume 188 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[118] František Simančík. Elimination of complex RIAs without automata. In Yevgeny Kazakov, Domenico Lembo, and Frank Wolter, editors, *Proc. 25th Int. Workshop on Description Logics (DL'12)*, volume 846 of *CEUR Workshop Proceedings*, pages 334–344. CEUR-WS.org, 2012.

[119] František Simančík and Andrew Bate. Consequence-based reasoning for $\mathcal{SHIQ}$. Technical report, University of Oxford, 2013. Available at `http://www.cs.ox.ac.uk/isg/people/frantisek.simancik/`.

[120] František Simančík, Yevgeny Kazakov, and Ian Horrocks. Consequence-based reasoning beyond Horn ontologies. In Toby Walsh, editor, *Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11)*, pages 1093–1098. AAAI Press/IJCAI, 2011.

[121] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *J. of Web Semantics*, 5(2):51–53, 2007.

[122] Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard, William Bug, Werner Ceusters, Louis J. Goldberg, Karen Eilbeck, Amelia Ireland, Christopher J. Mungall, The OBI Consortium, Neocles Leontis, Philippe Rocca-Serra, Alan Ruttenberg, Susanna-Assunta Sansone, Richard H. Scheuermann, Nigam Shah, Patricia L. Whetzeland, and Suzanna Lewis. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology*, 25:1251–1255, 2007.

[123] Dagobert Soergel, Boris Lauser, Anita C. Liang, Frehiwot Fisseha, Johannes Keizer, and Stephen Katz. Reengineering thesauri for new applications: The AGROVOC example. *J. of Digital Information*, 4(4), 2004.

[124] Ramakrishna Soma and Viktor K. Prasanna. Parallel inferencing for OWL knowledge bases. In *Proc. Int. Conf. on Parallel Processing (ICPP'08)*, pages 75–82. IEEE Computer Society, 2008.

[125] Boontawee Suntisrivaraporn, Franz Baader, Stefan Schulz, and Kent A. Spackman. Replacing SEP-triplets in SNOMED CT using tractable description logic operators. In *AIME*, volume 4594 of *LNCS*, pages 287–291. Springer, 2007.

[126] Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In Enrico Giunchiglia and Armando Tacchella, editors, *Proc. 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2003), Selected Revised Papers*, volume 2919 of *LNCS*, pages 188–202. Springer, 2003.

[127] Wei Tai, John Keeney, and Declan O'Sullivan. COROR: A composable rule-entailment OWL reasoner for resource-constrained devices. In Nick Bassiliades, Guido Governatori, and Adrian Paschke, editors, *Proc. 5th Int. Conf. on Rule-Based Reasoning, Programming, and Applications (RuleML Europe'11)*, volume 6826 of *LNCS*, pages 212–226. Springer, 2011.

[128] Herman J. ter Horst. 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.

[129] The Gene Ontology Consortium. Gene ontology annotations and resources. *Nucleic Acids Res*, 2012.

[130] Edward Thomas, Jeff Z. Pan, and Yuan Ren. TrOWL: Tractable OWL 2 reasoning infrastructure. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *Proc. 7th Extended Semantic Web Conf. (ESWC'10)*, volume 6089 of *LNCS*, pages 431–435. Springer, 2010.

[131] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, Germany, 2001.

[132] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *Proc. 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06)*, volume 4130 of *LNCS*, pages 292–297. Springer, 2006.

[133] Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider. Optimizing terminological reasoning for expressive description logics. *J. of Automated Reasoning*, 39(3):277–316, 2007.

[134] Dmitry Tsarkov and Ignazio Palmisano. Chainsaw: a metareasoner for large ontologies. In Ian Horrocks, Mikalai Yatskevich, and Ernesto Jimenez-Ruiz, editors, *Proc. OWL Reasoner Evaluation Workshop 2012 (ORE'12)*, volume 858 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[135] Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri Bal. WebPIE: a Web-scale parallel inference engine using MapReduce. *J. of Web Semantics*, pages 59–75, 2012.

[136] Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Scalable distributed reasoning using MapReduce. In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *Proc. 8th Int. Semantic Web Conf. (ISWC'09)*, volume 5823 of *LNCS*, pages 634–649. Springer, 2009.

[137] Jesse Weaver and James A. Hendler. Parallel materialization of the finite RDFS closure for hundreds of millions of triples. In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *Proc. 8th Int. Semantic Web Conf. (ISWC'09)*, volume 5823 of *LNCS*, pages 682–697. Springer, 2009.

[138] Christoph Weidenbach. Combining superposition, sorts and splitting. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, volume II, chapter 27, pages 1965–2013. Elsevier and MIT Press, 2001.

[139] Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning: Introduction and Applications*. McGraw-Hill, Inc., New York, NY, USA, second edition, 1992.

[140] Kejia Wu and Volker Haarslev. A parallel reasoner for the description logic $\mathcal{ALC}$. In Yevgeny Kazakov, Domenico Lembo, and Frank Wolter, editors, *Proc. 25th Int. Workshop on Description Logics (DL'12)*, volume 846 of *CEUR Workshop Proceedings*, pages 378–388. CEUR-WS.org, 2012.

[141] Z. Xiang, C. Mungall, A. Ruttenberg, and Y. He. Ontobee: A linked data server and browser for ontology terms. In *International Conference on Biomedical Ontologies (ICBO)*, pages 279–281, 2011.

[142] Guohui Xiao, Stijn Heymans, and Thomas Eiter. DReW: a reasoner for Datalog-rewritable description logics and dl-programs. In Thomas Eiter, Adil El Ghali, Sergio Fernández, Stijn Heymans, Thomas Krennwallner, and François Lévy, editors, *Proc. 1st Int. Workshop on Business Models, Business Rules and Ontologies (BuRO'10)*, pages 1–14. ONTORULE Project, 2010.