Journal of Artificial Intelligence Research 63 (2018) 625-690 Submitted 12/17; published 11/18

Consequence-Based Reasoning for Description Logics with Disjunctions and Number Restrictions

Andrew Bate ANDREW.BATE@CS.OX.AC.UK
Boris Motik BORIS.MOTIK@CS.OX.AC.UK
Bernardo Cuenca Grau BERNARDO.CUENCA.GRAU@CS.OX.AC.UK
David Tena Cucala DAVID.TENA.CUCALA@CS.OX.AC.UK
František Simančík FRANTISEK.SIMANCIK@CS.OX.AC.UK
Ian Horrocks IAN.HORROCKS@CS.OX.AC.UK
Department of Computer Science
University of Oxford
Wolfson Building, Parks Road
Oxford, OX1 3QD, United Kingdom

Abstract

Classification of description logic (DL) ontologies is a key computational problem in modern data management applications, so considerable effort has been devoted to the development and optimisation of practical reasoning calculi. Consequence-based calculi combine ideas from hypertableau and resolution in a way that has proved very effective in practice. However, existing consequence-based calculi can handle either Horn DLs (which do not support disjunction) or DLs without number restrictions. In this paper, we overcome this important limitation and present the first consequence-based calculus for deciding concept subsumption in the DL $\mathcal{ALCHIQ}^+$. Our calculus runs in exponential time assuming unary coding of numbers, and on $\mathcal{ELH}$ ontologies it runs in polynomial time. The extension to disjunctions and number restrictions is technically involved: we capture the relevant consequences using first-order clauses, and our inference rules adapt paramodulation techniques from first-order theorem proving. By using a well-known preprocessing step, the calculus can also decide concept subsumptions in $\mathcal{SRIQ}$—a rich DL that covers all features of OWL 2 DL apart from nominals and datatypes. We have implemented our calculus in a new reasoner called Sequoia. We present the architecture of our reasoner and discuss several novel and important implementation techniques such as clause indexing and redundancy elimination. Finally, we present the results of an extensive performance evaluation, which revealed Sequoia to be competitive with existing reasoners. Thus, the calculus and the techniques we present in this paper provide an important addition to the repertoire of practical implementation techniques for description logic reasoning.

1. Introduction

Description logics (DLs) (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2003) are a prominent family of knowledge representation formalisms. They can describe key entities of a domain of interest using *concepts* (i.e., unary predicates) and the relationships between entities using *roles* (i.e., binary predicates). An *ontology* is a formal description of a domain of interest, and it consists of first-order sentences constructed from concepts and roles. *Subsumption* is the problem of determining whether each instance of one concept is also an instance of another concept in all models of an ontology, and *ontology satisfiability* is

the problem of determining whether an ontology admits a model. For expressive DLs, both problems are interreducible and are of high worst-case complexity, ranging from ExpTime (Sattler & Vardi, 2001) up to N2ExpTime (Kazakov, 2008). DLs provide the logical foundation for the OWL 2 DL ontology language, which is used nowadays in numerous practical applications. Consequently, many different DL reasoning calculi have been developed since the 1980s, and they provide the basis for several practically effective reasoners.

Tableau calculi (Baader & Sattler, 2001) comprise a prominent group of reasoning calculi that try to construct a finite representation of a canonical model of the ontology disproving a postulated subsumption. Various blocking techniques (Motik, Shearer, & Horrocks, 2009) are used to prevent infinite model expansion and thus ensure termination. Traditional tableau calculi are not worst-case optimal, but they incorporate numerous optimisations such as absorption that facilitate processing of large ontologies expressed in rich DL languages. Such calculi provide the foundation for several widely known reasoners such as FaCT++ (Tsarkov & Horrocks, 2006), Pellet (Sirin, Parsia, Cuenca Grau, Kalyanpur, & Katz, 2007), and RacerPro (Haarslev, Hidde, Möller, & Wessel, 2012). Analogously to the hypertableau calculus for first-order logic (Baumgartner, Furbach, & Niemelä, 1996), the hypertableau calculus for DLs (Motik et al., 2009) incorporates many optimisations into its core that reduce don't-know nondeterminism during reasoning, and it provides the foundation for the HermiT reasoner (Glimm, Horrocks, Motik, Stoilos, & Wang, 2014). The aforementioned systems can successfully process numerous ontologies, but sometimes they may construct very large model representations, which is a source of performance problems. Such problems are further exacerbated by the large number of subsumption tests often required to classify an ontology. More recently, tableau calculi have been extended with global caching techniques that ensure worst-case optimal behaviour (Goré & Nguyen, 2007, 2013; Nguyen, 2013; Nguyen & Golińska-Pilarek, 2014; Nguyen, 2014). To the best of our knowledge, such techniques have been implemented only in prototypes such as TGC2.[1]

Another large group of reasoning calculi for DLs and their extensions are based on parameterisations of first-order resolution (Bachmair & Ganzinger, 2001). Such procedures can be broadly subdivided into two subgroups. The first subgroup consists of procedures that essentially simulate (hyper)tableau techniques in a resolution framework; Hustadt and Schmidt (1999) call this *decidability by selection*. To ensure termination, these techniques either restrict the ontology so that model construction terminates naturally (e.g., by disallowing terminological cycles), or they stop model construction using techniques analogous to blocking (Georgieva, Hustadt, & Schmidt, 2003). The second subgroup consists of procedures that, rather than constructing finite model representations, parameterise resolution in a way that ensures that the calculus can derive only a finite number of clauses given a finite signature; Hustadt and Schmidt (1999) call this *decidability by ordering*. Such approaches can usually handle ontologies with terminological cycles, and they have been developed for DLs such as $\mathcal{ALB}$ (de Nivelle, Schmidt, & Hustadt, 2000; Hustadt & Schmidt, 2002; Schmidt & Hustadt, 2013) and $\mathcal{SHIQ}$ (Hustadt, Motik, & Sattler, 2008), and their generalisations such as the guarded fragment (Ganzinger & de Nivelle, 1999). The KAON2 reasoner (Hustadt et al., 2008) implements such a technique for the DL $\mathcal{SHIQ}$.

1. `http://www.mimuw.edu.pl/~nguyen/TGC2/index.html`

A breakthrough in DL reasoning came in the form of *consequence-based calculi*. The polynomial-time reasoning algorithm by Baader, Brandt, and Lutz (2005) for the lightweight DL $\mathcal{EL}$ can be seen as the first such calculus. This algorithm was later significantly improved and extended to the DL Horn-$\mathcal{SHIQ}$ (Kazakov, 2009), which allowed the CB reasoner to be the first to classify the full version of the GALEN ontology (Solomon, Roberts, Rogers, C.J., & Rector, 2000) and thus solve a long-standing open problem in DL reasoning. This calculus was later extended to the DLs $\mathcal{ALCH}$ (Simančík, Kazakov, & Horrocks, 2011) and $\mathcal{ALCI}$ (Simančík, Motik, & Horrocks, 2014) that support concept disjunction, but not number restrictions. Recently, Vlasenko, Daryalal, Haarslev, and Jaumard (2016) presented a calculus for the DL $\mathcal{ELQ}$ that combines consequence-based reasoning with integer linear programming in order to efficiently reason with number restrictions, and Karahroodi and Haarslev (2017) presented a similar approach for the DL $\mathcal{SHOQ}$. All of these calculi can intuitively be seen as combining ideas from resolution and hypertableau (see Section 3): as in resolution, they systematically derive certain consequences in order to describe a class of ontology models that can be constructed in a specific way from a saturated set of consequences; and as in (hyper)tableau, they construct in a goal-directed fashion an outline of an ontology model. They are also not only refutationally complete, but can check several subsumptions in a single run, which can considerably reduce the overall work for higher-level reasoning tasks such as ontology classification. Finally, similarly to resolution-based decision procedures, consequence-based calculi are typically worst-case optimal.

Although many consequence-based calculi are known, none of them can handle DLs with both disjunctions and number restrictions. This drawback has so far been addressed using hybrid reasoning approaches. The Konclude reasoner (Steigmiller, Liebig, & Glimm, 2014) is based on a tableau calculus, but it also uses a sound but incomplete consequence-based calculus to speed up the derivation of certain consequences. The MORe reasoner (Armas Romero, Cuenca Grau, & Horrocks, 2012) classifies an ontology by extracting a fragment of the ontology that can be processed more efficiently with a consequence-based calculus and delegating the rest of the ontology to a hypertableau reasoner. While such combinations often perform well in practice, they are not worst-case optimal. Hence, developing worst-case optimal consequence-based calculi that can handle expressive DLs remains an open problem that is interesting from both a practical and a theoretical perspective.

As we discuss in detail in Section 3, handling both disjunctions and number restrictions poses a key new challenge: number restrictions require equality reasoning, which, together with disjunctions, can impose complex constraints on an ontology model; however, it is unclear whether such constraints can be captured using the syntax of DLs themselves. This is different from the known consequence-based calculi, where the derived consequences can be represented as DL axioms. Hence, extending a framework such as the one by Simančík et al. (2014) to number restrictions and disjunctions is technically challenging.

In Section 4 we present the first consequence-based calculus for subsumption in the DL $\mathcal{ALCHIQ}^{+}$, which supports all Boolean connectives on concepts, self-restrictions, number restrictions, role hierarchies, and inverse and reflexive roles; we do not consider ABoxes since we focus on subsumption. Following the ideas from the aforementioned resolution-based decision procedures, we encode the calculus' consequences as first-order clauses of a specific form, and we handle equality using a variant of *ordered paramodulation* (Nieuwenhuis & Rubio, 1995; Bachmair & Ganzinger, 1998)—a state-of-the-art calculus for equational theo-

rem proving used in modern theorem provers such as SPASS (Weidenbach, Gaede, & Rock, 1996), E (Schulz, 2002), and Vampire (Riazanov & Voronkov, 2002). Our calculus runs in exponential time assuming unary coding of numbers. By applying the encoding of role inclusion axioms (Demri & de Nivelle, 2005; Schmidt & Hustadt, 2007; Simančík, 2012) in a preprocessing step, our calculus can also handle $\mathcal{SRIQ}$, which covers all of OWL 2 DL except for nominals and datatypes. On $\mathcal{ELH}$ (Baader et al., 2005) ontologies our calculus runs in polynomial time; moreover, we have carefully constrained the inference rules so that, on $\mathcal{EL}$ ontologies, the calculus mimics existing calculi and thus exhibits pay-as-you-go behaviour. As a result, one can expect good performance on 'mostly-$\mathcal{EL}$' ontologies.

We have implemented our calculus in a new reasoner called *Sequoia*.[2] In Section 5 we present the system's architecture and discuss several techniques that we found critical for performance, such as clause indexing and implementing the saturation process. We also discuss redundancy elimination techniques that can simplify or even eliminate certain clauses from saturation. Such techniques play a key role in of first-order theorem proving (Riazanov & Voronkov, 2003; Hustadt, Motik, & Sattler, 2007; Schulz, 2004; Bachmair & Ganzinger, 2001), and we found them equally important in DL reasoning as well.

In Section 6 we present the results of our performance evaluation in which we compared Sequoia with FaCT++, Pellet, HermiT, and Konclude on classifying a corpus consisting of 777 description logic ontologies. Sequoia typically outperforms FaCT++, Pellet, and HermiT and it often exhibits comparable performance to Konclude. Moreover, we have also compared Sequoia with $\mathcal{EL}$ reasoners jcel (Mendez, 2012), Snorocket (Metke-Jimenez & Lawley, 2013), and ELK (Kazakov, Krötzsch, & Simančík, 2014) on the appropriate subset of the test ontologies. Even though it implements a more complex logic, the performance of Sequoia was comparable to Snorocket and ELK, which we take as empirical confirmation of the pay-as-you-go property. Thus, the techniques we present in this paper provide an important addition to the repertoire of practical implementation techniques for DLs.

This is an extension of our earlier work published at the KR 2016 conference (Bate, Motik, Cuenca Grau, Simančík, & Horrocks, 2016). New material includes the addition of self-reflexivity (which considerably changes the calculus and the completeness proofs), a description of Sequoia's architecture and the relevant implementation techniques (see Section 5), and an updated evaluation and comparison with $\mathcal{EL}$ reasoners (see Section 6).

2. Preliminaries

In this section we recall well-known definitions of many-sorted clausal equational logic (Section 2.1), encoding of predicates by function symbols in an equational theory (Section 2.2), term orders (Section 2.3), and description logics (Section 2.4).

2.1 Many-Sorted Clausal Equational Logic

We first recall the definitions of many-sorted clausal equational logic (Walther, 1987). We assume in our presentation that equality is the only predicate. This is without loss of generality since it is well known that one can encode ordinary atoms using only the equality predicate (Nieuwenhuis & Rubio, 2001); we discuss this in detail in Section 2.2.

2. `http://www.cs.ox.ac.uk/isg/tools/Sequoia/`

First-order clauses are constructed using the symbols of a *many-sorted signature*, which is a pair $\Sigma = \left(\Sigma^O, \Sigma^F\right)$ where Σ^O is a nonempty set of *sorts*, and Σ^F is a countable set of function symbols. Each function symbol $f \in \Sigma^F$ is associated with a *symbol type*, which is an expression of the form $o_1 \times \cdots \times o_n \to o$ where $n \geq 0$ and $\{o_1, \ldots, o_n, o\} \subseteq \Sigma^O$ are sorts; such f is *of arity* n and *of sort* o, and it is a *constant* if n is zero. Given a sort o, set Σ^F_o contains precisely all symbols in Σ^F of sort o. We assume that, for each sort $o \in \Sigma^O$, there exists a distinct, countable set of *variables* of sort o. The set of *terms*, each associated with a sort in Σ^O, is the smallest set such that (i) each variable x of sort o is a term of sort o, and (ii) for each function symbol $f \in \Sigma^F$ with symbol type $o_1 \times \cdots \times o_n \to o$ and for all terms $t_1, \ldots, t_n$ where t_i is of sort o_i for $i \leq i \leq n$, expression $f(t_1, \ldots, t_n)$ is a term of sort o. A term of sort o is also called an *o-term*. Finally, the signature Σ must ensure that, for each sort $o \in \Sigma^O$, there exists at least one variable-free term of sort o.

An *equality* is an expression of the form $t_1 \approx t_2$ where t_1 and t_2 are terms of the same sort. An *inequality* is the negation of an equality $\neg(t_1 \approx t_2)$, and it is typically written as $t_1 \not\approx t_2$. We assume that $\approx$ and $\not\approx$ are implicitly symmetric—that is, for $\bowtie \in \{\approx, \not\approx\}$, expressions $t_1 \bowtie t_2$ and $t_2 \bowtie t_1$ are one and the same. A *literal* is an equality or an inequality. A *clause* is a formula of the form $\forall \vec{x}.[\Gamma \to \Delta]$ where Γ is a conjunction of equalities called the *body*, Δ is a disjunction of literals called the *head*, and $\vec{x}$ contains all variables occurring in the clause; quantifier $\forall \vec{x}$ is usually left implicit. It is common practice to assume that clause heads contain equalities only; however, in this paper it will be convenient to consider clauses whose heads contain both equalities and inequalities since this will be needed for the inference rules of our calculus. We often treat conjunctions and disjunctions as sets (i.e., they are unordered and without repetition) and use them in standard set operations. We write the empty conjunction (resp. disjunction) as $\top$ (resp. $\bot$). A term, literal, clause, or a set thereof is *ground* if it does not contain a variable.

A *substitution* σ is a mapping of variables to terms such that (i) for each variable x, the sort of x and $\sigma(x)$ are the same, and (ii) the set of variables $\{\, x \mid \sigma(x) \neq x \,\}$ is finite. We often write substitutions as $\sigma = \{x_1 \mapsto t_1,\ \ldots,\ x_k \mapsto t_k\}$—that is, we list all nonidentity mappings. The result of applying a substitution σ to a term or a formula α, written $\alpha\sigma$, is obtained by replacing in α each free occurrence of a variable x with $\sigma(x)$. The *domain* of σ is the set containing each variable x such that $\sigma(x) \neq x$. Moreover, σ is *ground* if $\sigma(x)$ is ground for each variable x in the domain of σ.

A *position* is a finite sequence of positive integers and it is written $i_1.i_2 \ldots i_n$; the empty sequence is written ϵ. Let p be a position and let t be a term. The *subterm of t at position p* is defined inductively as follows: $t|_\epsilon = t$ and, if $t = f(t_1, \ldots, t_n)$, then $t|_{i.p} = t_i|_p$ if $1 \leq i \leq n$ (and it is undefined if $i > n$). A position p is *proper* in a term t if $t|_p \neq t$. Finally, $s[t]_p$ is the term obtained by replacing the subterm of s at position p with t, provided that terms $s|_p$ and t are of the same sort (and it is undefined otherwise).

Clauses are commonly interpreted using Herbrand interpretations, where each constant is interpreted by itself and an application of a function symbol f to ground arguments $t_1, \ldots, t_n$ is interpreted as the ground term $f(t_1, \ldots, t_n)$. Formally, the *Herbrand universe* for Σ is the set HU of all ground terms constructed using the symbols of Σ (while respecting the sort restrictions). A *Herbrand equality interpretation* I is a set of ground equalities that satisfies the usual properties of equality—that is, $\approx$ is reflexive, symmetric, and transitive in I, $\{s_1 \approx t_1, \ldots, s_n \approx t_n\} \subseteq I$ implies $f(s_1, \ldots, s_n) \approx f(t_1, \ldots, t_n) \in I$ for

each function symbol f and terms $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$ in HU of appropriate sorts, and $\{s \approx t, s \approx s', t \approx t'\} \subseteq I$ implies $s' \approx t' \in I$ for all terms s, s', t, and t' in HU. For α a ground conjunction, ground disjunction, (not necessarily ground) clause, or a set thereof, satisfaction of α in a Herbrand equality interpretation I, written $I \models \alpha$, is defined as usual, but with the difference that each variable of sort o quantifies only over the terms in HU of sort o (instead of quantifying over all of HU). Note that a ground disjunction of literals Δ may contain inequalities so $I \models \Delta$ does not necessarily imply $I \cap \Delta \neq \emptyset$. Let $\mathcal{O}$ be a set of clauses. A Herbrand equality interpretation I *satisfies* $\mathcal{O}$ (i.e., I is a *model* of $\mathcal{O}$) if $I \models \mathcal{O}$; moreover, $\mathcal{O}$ is *satisfiable* if it admits a model; finally, $\mathcal{O}$ *entails* a clause $\Gamma \to \Delta$ if $I \models \Gamma \to \Delta$ holds for each model I of $\mathcal{O}$.

To understand how the presence of sorts affects the semantics of first-order logic, let $\mathcal{O}$ be the set containing clauses $\to x_1 \approx x_2$ and $\to c_1 \not\approx c_2$. Set $\mathcal{O}$ is unsatisfiable if variables x_1 and x_2 are of the same sort as constants c_1 and c_2; otherwise, $\mathcal{O}$ is satisfiable because variables x_1 and x_2 then do not range over the constants c_1 and c_2.

2.2 Encoding Predicates by Function Symbols

Herbrand equality interpretations can be conveniently represented using rewrite systems (see Appendix B), in which case it is convenient to assume that equality is the only predicate. This is without loss of generality since, as we discuss next, each formula containing predicates other than equality can be transformed into an equisatisfiable formula of multi-sorted first-order logic where equality is the only predicate.

In the rest of this paper, we assume that $\mathsf{p} \in \Sigma^O$ is a distinct *predicate* sort whose use is restricted such that, for each type $o_1 \times \cdots \times o_n \to \mathsf{p}$ of a symbol in the signature Σ, we have $o_i \neq \mathsf{p}$ for each $1 \leq i \leq n$. We also assume that true is a distinct constant of sort p. Then, an atom of the form $P(t_1, \ldots, t_n)$ where P is a predicate other than equality is transformed into $P(t_1, \ldots, t_n) \approx \mathsf{true}$, where P is a function symbol of sort p. For example, given a function symbol P of sort p and a function symbol f of a different sort, both $f(P(x))$ and $P(P(x))$ are not well-formed, whereas $P(f(x))$ is a well-formed p-term. It is well known that this transformation preserves formula satisfiability and entailment (Nieuwenhuis & Rubio, 2001). Whenever the intended meaning is clear from the context, we abbreviate $P(t_1, \ldots, t_n) \approx \mathsf{true}$ as just $P(t_1, \ldots, t_n)$, and we call an equality of that form an *atom*.

2.3 Term and Literal Orders

A *strict order* $\succ$ on a domain set Ω is an irreflexive, asymmetric, and transitive relation on Ω; and $\succeq$ is the *partial order* induced by $\succ$. A strict order $\succ$ is *total* if, for all $a, b \in \Omega$, we have $a \succ b$, $b \succ a$, or $a = b$. Given an element $b \in \Omega$, a subset $S \subseteq \Omega$, and for $\circ$ equal to $\succ$ of $\succeq$, the notation $S \circ b$ abbreviates $\exists a \in S : a \circ b$.

A *multiset* M on a domain set Ω is a function assigning to each element $a \in \Omega$ a non-negative integer $M(a)$. Multiset M is finite if the set $\{\, a \in \Omega \mid M(a) \neq 0 \,\}$ is finite, and we often write such multisets as sets where an element a is repeated $M(a)$ times. The difference $M \setminus N$ of multisets M and N on Ω is defined by $(M \setminus N)(a) = \max(0, M(a) - N(a))$ for each element $a \in \Omega$. The *multiset extension* $\succ_{mul}$ of an order $\succ$ on Ω is the order defined on all finite multisets M and N over Ω as follows: $M \succ_{mul} N$ if and only if $M \neq N$ and, for each $n \in N \setminus M$, there exists $m \in M \setminus N$ such that $m \succ n$.

Table 1: Translating Normalised $\mathcal{ALCHIQ}^+$ Ontologies into DL-Clauses

DL1	$\prod_{1\leq i\leq n} B_i \sqsubseteq \bigsqcup_{n+1\leq i\leq m} B_i$	$\rightsquigarrow$	$\bigwedge_{1\leq i\leq n} B_i(x) \rightarrow \bigvee_{n+1\leq i\leq m} B_i(x)$	
DL2	$B_1 \sqsubseteq \geqslant n\, S.B_2$	$\rightsquigarrow$	$B_1(x) \rightarrow S(x, f_i(x))$ $B_1(x) \rightarrow B_2(f_i(x))$ $B_1(x) \rightarrow f_i(x) \not\approx f_j(x)$	for $1 \leq i \leq n$ for $1 \leq i \leq n$ for $1 \leq i < j \leq n$
DL3	$\exists S.B_1 \sqsubseteq B_2$	$\rightsquigarrow$	$S(z_1, x) \wedge B_1(x) \rightarrow B_2(z_1)$	
DL4	$B_1 \sqsubseteq \leqslant n\, S.B_2$	$\rightsquigarrow$	$S(z_1, x) \wedge B_2(x) \rightarrow S_{B_2}(z_1, x)$ $B_1(x) \wedge \bigwedge_{1\leq i\leq n+1} S_{B_2}(x, z_i) \rightarrow \bigvee_{1\leq i<j\leq n+1} z_i \approx z_j$	S_{B_2} is fresh and unique for S and B_2
DL5	$B \sqsubseteq \exists S.\mathsf{Self}$	$\rightsquigarrow$	$B(x) \rightarrow S(x, x)$	
DL6	$\exists S.\mathsf{Self} \sqsubseteq B$	$\rightsquigarrow$	$S(x, x) \rightarrow B(x)$	
DL7	$S_1 \sqsubseteq S_2$	$\rightsquigarrow$	$S_1(z_1, x) \rightarrow S_2(z_1, x)$	
DL8	$S_1 \sqsubseteq S_2^-$	$\rightsquigarrow$	$S_1(z_1, x) \rightarrow S_2(x, z_1)$	
DL9	$\mathsf{Disjoint}(S_1, S_2)$	$\rightsquigarrow$	$S_1(z_1, x) \wedge S_2(z_2, x) \rightarrow \bot$	

Note: $B_{(i)}$ are atomic concepts, $S_{(i)}$ are atomic roles, and n is a nonnegative integer.

A *term order* $\succ$ is a *strict* order on the set of all terms. As usual in resolution-based equational theorem proving (Nieuwenhuis & Rubio, 2001), we extend $\succ$ to literals by identifying each $s \not\approx t$ with the multiset $\{s, s, t, t\}$ and each $s \approx t$ with the multiset $\{s, t\}$, and by comparing the result using the multiset extension of $\succ$. We reuse the symbol $\succ$ for the induced literal order since the intended meaning should be clear from the context.

2.4 Description Logics

We next introduce the description logic $\mathcal{ALCHIQ}^+$ that we consider in this paper. A DL signature consists of *atomic concepts* and *atomic roles*. A *role* is an expression of the form S or S^-, for S an atomic role. The set of *concepts* is inductively defined as containing all atomic concepts, $\top$ and $\bot$, and concepts $\neg C$, $C_1 \sqcap C_2$, $C_1 \sqcup C_2$, $\exists R.C$, $\forall R.C$, $\exists R.\mathsf{Self}$, $\geqslant n\, R.C$, and $\leqslant n\, R.C$ for R a role, C, C_1, and C_2 concepts, and n a nonnegative integer. An *ontology* is a finite set of axioms of the form $C_1 \sqsubseteq C_2$, $R_1 \sqsubseteq R_2$, or $\mathsf{Disjoint}(R_1, R_2)$, where C_i are concepts and R_i are roles. Using a *normalisation* process analogous to the *structural transformation* (Nonnengart & Weidenbach, 2001) from first-order theorem proving, each ontology can be transformed in polynomial time into a *normalised* ontology containing axioms only of the form shown on the left-hand side of Table 1 while preserving all ontology consequences; this process is well understood (Motik et al., 2009; Hustadt & Schmidt, 2002) so we do not discuss it further. Ontologies are interpreted using a first-order semantics.

Our DL reasoning calculus is based on resolution, so it requires ontologies to be represented using first-order clauses. Thus, we next redefine the notions of DL axioms and ontologies in a way that is more suitable to our work. Ontologies are constructed using a signature $\Sigma = \left(\Sigma^O, \Sigma^F\right)$ where Σ^O contains the predicate sort p and the *abstract sort* a. Set Σ^F contains *atomic concepts* B_i of type $\mathsf{a} \rightarrow \mathsf{p}$, *atomic roles* S_i of type $\mathsf{a} \times \mathsf{a} \rightarrow \mathsf{p}$, and

successor functions f_j of type $\mathsf{a} \to \mathsf{a}$. A term of the form $f_j(t)$ is an *f_j-successor* of t, and t is the *predecessor* of $f_j(t)$. We distinguish a distinct *central variable* x from the remaining *neighbour variables* z_j, all of which are of sort a. A *DL-a-term* is of the form z_j, x, or $f_j(x)$. A *DL-p-term* is of the form $B_i(z_j)$, $B_i(x)$, $B_i(f_j(x))$, $S_i(z_j, x)$, $S_i(x, z_j)$, $S_i(x, x)$, $S_i(f_j(x), x)$, or $S_i(x, f_j(x))$. A *DL-literal* is either an atom of the form $A \approx \mathsf{true}$ where A is a DL-p-term (which we often abbreviate as just A), or of the form $l \bowtie r$ where l and r are DL-a-terms and $\bowtie \in \{\approx, \not\approx\}$. A *DL-clause* contains only atoms of the form $B_i(x)$, $S_i(x, x)$, $S_i(z_j, x)$, and $S_i(x, z_j)$ in the body and only DL-literals in the head, and each variable z_j occurring in the clause also occurs in the body. Unless stated otherwise, in the rest of this paper we assume that an *ontology* $\mathcal{O}$ is given as a finite set of DL-clauses.

In this paper we consider the problem of deciding whether $\mathcal{O} \models \Gamma \to \Delta$ holds for a given ontology $\mathcal{O}$ and a *query clause* $\Gamma \to \Delta$—that is, a DL-clause where all literals are atoms of the form $B_i(x)$ or $S_i(x, x)$. We can also check whether $\mathcal{O}$ is inconsistent by checking whether $\mathcal{O} \models \top \to \bot$ holds. This provides us with the basic building block for implementing ontology classification as outlined in Section 4.3.

Each normalised $\mathcal{ALCHIQ}^+$ axiom can be transformed into DL-clauses as shown in Table 1. This step is polynomial if numbers are coded in unary, and it is exponential otherwise. It uses the well-known correspondence between DL axioms and clauses (Schmidt & Hustadt, 2007): an axiom is first translated into a first-order sentence (Baader et al., 2003), existential quantifiers (if any) are skolemised, and the result is converted into clauses using de Morgan laws. For $B_1 \sqsubseteq \leqslant n\, S.B_2$, the standard translation produces

$$B_1(x) \wedge \bigwedge_{1 \leq i \leq n+1} \Big[S(x, z_i) \wedge B_2(z_i) \Big] \to \bigvee_{1 \leq i < j \leq n+1} z_i \approx z_j. \tag{1}$$

Our calculus, however, requires atoms of the form $B_2(z_i)$ to not occur in clause bodies. To transform (1) to a DL-clause, we introduce a fresh role S_{B_2} uniquely associated with S and B_2 and apply the structural transformation $S(x, z_1) \wedge B_2(z_1)$: we introduce a DL-clause $S(z_1, x) \wedge B_2(x) \to S_{B_2}(z_1, x)$ and replace each $S(x, z_i) \wedge B_2(z_i)$ in (1) with $S_{B_2}(x, z_i)$.

The description logic $\mathcal{ELH}$ (Baader et al., 2005) is the restriction of $\mathcal{ALCHIQ}^+$ where all roles are required to be atomic, and all concepts are required to be of the form B, $C_1 \sqcap C_2$, $\exists S.C$, $\bot$, or $\top$. Normalised $\mathcal{ELH}$ axioms are of type DL1 with $n \leq m \leq n+1$, DL2 with $n = 1$ and DL3, and DL7 from Table 1, and so an $\mathcal{ELH}$ ontology $\mathcal{O}$ contains DL-clauses of the corresponding types. Moreover, the description logic $\mathcal{EL}$ is obtained from $\mathcal{ELH}$ by disallowing axioms of the form DL7.

The DL $\mathcal{SRIQ}$ (Horrocks & Sattler, 2004) extends $\mathcal{ALCHIQ}^+$ with *role inclusion axioms* of the form $S_1 \circ \cdots \circ S_n \sqsubseteq S$, where $S_{(i)}$ are roles. Unrestricted use of such axioms leads to undecidability, but, under certain global restrictions, such axioms can be encoded by DL-clauses without affecting the entailed query clauses (Demri & de Nivelle, 2005; Schmidt & Hustadt, 2007; Simančík, 2012). By combining this transformation with our algorithm, we obtain an optimal reasoning procedure for $\mathcal{SRIQ}$, assuming unary coding of numbers.

DL-clauses can capture axioms outside $\mathcal{ALCHIQ}^+$. For example, (2) captures relativized role inclusion, and (3) is a safe role expression (Tobies, 2001). As in other approaches that translate DL axioms into clauses (Hustadt et al., 2008; Motik et al., 2009), such DL-clauses can be included in ontologies and will be correctly handled by our algorithm.

$$S_1(z_1, x) \wedge B(x) \to S_2(z_1, x) \tag{2}$$

Ontology $\mathcal{O}_1$

$B_i \sqsubseteq \exists S_1.B_{i+1}$	$\rightsquigarrow$	$B_i(x) \rightarrow S_1(x, f_{i+1}(x))$	(4)	for $0 \leq i < n$
		$B_i(x) \rightarrow B_{i+1}(f_{i+1}(x))$	(5)	
$B_i \sqsubseteq \exists S_2.B_{i+1}$	$\rightsquigarrow$	$B_i(x) \rightarrow S_2(x, g_{i+1}(x))$	(6)	
		$B_i(x) \rightarrow B_{i+1}(g_{i+1}(x))$	(7)	
$B_n \sqsubseteq C_n$	$\rightsquigarrow$	$B_n(x) \rightarrow C_n(x)$	(8)	
$\exists S_1.C_{i+1} \sqsubseteq C_i$	$\rightsquigarrow$	$S_1(z_1, x) \wedge C_{i+1}(x) \rightarrow C_i(z_1)$	(9)	for $0 \leq i < n$
$\exists S_2.C_{i+1} \sqsubseteq C_i$	$\rightsquigarrow$	$S_2(z_1, x) \wedge C_{i+1}(x) \rightarrow C_i(z_1)$	(10)	
$C_0 \sqcap \cdots \sqcap C_n \sqsubseteq \bot$	$\rightsquigarrow$	$C_0(x) \wedge \cdots \wedge C_n(x) \rightarrow \bot$	(11)	

Figure 1: Example Ontology $\mathcal{O}_1$

$$S_1(x, z_1) \wedge S_2(z_1, x) \rightarrow S_3(x, z_1) \vee S_4(z_1, x) \tag{3}$$

3. Motivation

In this section we motivate consequence-based calculi and present an outline of our approach. In Section 3.1 we discuss certain inefficiencies of existing DL calculi. In Section 3.2 we discuss the main ideas of consequence-based reasoning. Finally, in Section 3.3 we discuss the difficulties in extending the framework to disjunctions and number restrictions.

3.1 Why Consequence-Based Calculi?

Let $\mathcal{O}_1$ be the $\mathcal{EL}$ ontology shown in Figure 1; one can check that $\mathcal{O}_1 \not\models B_i(x) \rightarrow \bot$ holds for $0 \leq i \leq n$. We next discuss how both (hyper)tableau and resolution—DL calculi commonly used in practice—draw irrelevant conclusions while proving this.

To prove $\mathcal{O}_1 \not\models B_0(x) \rightarrow \bot$, a (hyper)tableau calculus tries to construct a model of $\mathcal{O}$ and $B_0(a)$ by applying the DL-clauses (4)–(10) in a forward-chaining manner. Specifically, DL-clauses (4)–(8) construct a tree-shaped model of depth n and a fanout of two where nodes at depth i are labelled by B_i, and then DL-clauses (8)–(10) ensure that nodes at depth i are also labelled by C_i. Forward chaining ensures that reasoning is goal-oriented. However, all nodes labelled with B_i are equivalent in the sense that they participate in exactly the same concepts and roles, revealing a weakness of (hyper)tableau calculi: the constructed models can be large (exponential in our example) and highly redundant, which leads to inefficiency in practice and often prevents (hyper)tableau calculi from being worst-case optimal. Since $\mathcal{O}_1$ is an $\mathcal{EL}$ ontology, multiple existential restrictions can satisfied using the same object, and so a polynomially-sized model can be constructed using the algorithm by (Baader et al., 2005); however, number restrictions, inverse roles, or disjunctions make such an approach unsound for expressive DLs such as $\mathcal{ALCHIQ}^+$. Techniques such as *caching* (Goré & Nguyen, 2007) or *anywhere blocking* (Motik et al., 2009) can sometimes constrain the model construction. Nevertheless, the resulting model representations tend to be large, which is a key problem for (hyper)tableau reasoners (Motik et al., 2009). Models can also be constructed in a resolution framework (Hustadt & Schmidt, 1999), but such procedures suffer from essentially the same drawbacks.

Instead of constructing a model explicitly, resolution can decide the satisfiability of a theory by deriving clauses that 'summarise' a model that we call *canonical* (i.e., a model that can be constructed in from a saturated set of clauses as described in Appendix C). As mentioned in Section 1, a worst-case optimal DL reasoning procedure can often be obtained by parameterising resolution so only finitely many clauses can be derived (de Nivelle et al., 2000; Hustadt & Schmidt, 2002; Schmidt & Hustadt, 2013; Hustadt et al., 2008). All such parameterisations we are aware of ensure that inferences take place only with atoms containing all variables of a clause, and that 'deepest' such atoms are preferred. On $\mathcal{O}_1$, regardless of the ordering, resolution derives (12) for each i with $0 \leq i \leq n$. Moreover, clause (11) leads to the derivation of clauses of the form (13) and (14); which of these clauses are derived depends on the exact ordering of the atoms on the clauses.

$$B_i(x) \to C_i(x) \tag{12}$$

$$C_0(f_i(x)) \land \cdots \land C_{i-1}(f_i(x)) \land B_{i-1}(x) \land C_{i+1}(f_i(x)) \land \cdots \land C_n(f_i(x)) \to \bot \tag{13}$$

$$C_0(g_i(x)) \land \cdots \land C_{i-1}(i_i(x)) \land B_{i-1}(x) \land C_{i+1}(g_i(x)) \land \cdots \land C_n(g_i(x)) \to \bot \tag{14}$$

Note, however, that clause (11) is irrelevant: we can construct in a goal-directed fashion (e.g., using hypertableau) a model of $\mathcal{O}_1$ in which no domain element participates in all C_i. All consequences of (11), such as (13) and (14), are thus irrelevant as well. Deriving such clauses can be problematic if $\mathcal{O}_1$ were extended with further axioms since each irrelevant clause can then produce further irrelevant clauses.

3.2 Main Ideas of Consequence-Based Reasoning

Consequence-based calculi addresses the issues from Section 3.1 by combining ideas from (hyper)tableau and resolution with the goal of restricting inferences. Although the technical details and the terminology vary considerably in the literature, we next identify four characteristics that, we believe, lie at the core of consequence-based reasoning.

The first characteristic of consequence-based calculi is that they are not just complete for refutation, but they actively derive all ontology consequences of a certain form. They usually target concept subsumption and can determine all relevant subsumption relationships in just one run of the algorithm, which greatly benefits the performance of ontology classification in practice. For example, our calculus can decide in a single run as many entailments of the form $\mathcal{O} \models \Gamma \to \Delta$ as required, for $\Gamma \to \Delta$ a query clause—that is, a clause where Γ and Δ consist of atoms of the form $B(x)$ and $S(x, x)$. Please note that $\top \to \bot$ is a query clause, so our calculus can also decide ontology satisfiability.

The second characteristic is closely connected to the first one: reasoning proceeds by deriving universally quantified consequences, rather than by explicitly constructing an ontology model. Many existing calculi represent the relevant consequences using DL-style axioms, possibly complemented by other axioms such as (non)emptiness constraints. For example, Kazakov (2009) represents Horn-$\mathcal{SHIQ}$ consequences as DL axioms of the form $M \sqsubseteq B$, $M \sqsubseteq \bot$, $M \sqsubseteq \forall R.B$, $M \sqsubseteq \leqslant 1\, R.B$, and $M \sqsubseteq \exists R.N$, where M and N are (possibly empty) conjunctions of atomic concepts, R is a role, and B is an atomic concepts. Similarly, Simančík et al. (2014) represent $\mathcal{ALCI}$ consequences as axioms of the form (15), where B_i, B_j, B_k, and B_ℓ are atomic concepts, and R_k and R_ℓ are (not necessarily atomic) roles:

$$\bigsqcap B_i \sqsubseteq \bigsqcup B_j \sqcup \bigsqcup \exists R_k.B_k \sqcup \bigsqcup \forall R_\ell.B_\ell \tag{15}$$

A similar approach was used for $\mathcal{ALCH}$ (Simančík et al., 2011). Baader et al. (2005) handle ontologies in extensions of $\mathcal{EL}$ by constructing a graph-like structure where nodes and edges are labelled with atomic concepts and atomic roles, respectively, but an S-labelled edge from node B_1 to node B_2 encodes a consequence $B_1 \sqsubseteq \exists S.B_2$. In contrast, the calculus by Kazakov et al. (2014) represents such consequences directly as DL-style axioms. Despite these differences, the consequences in all these approaches can succinctly describe a *canonical* ontology model: instead of describing identical model elements separately as is done in (hyper)tableau calculi, consequences are universally quantified so they describe all identical model elements 'at once'. This can often overcome the problem of constructing large repetitive models outlined in Section 3.1, and it is similar to resolution, which can prove the satisfiability of a theory without explicitly constructing a model.

The third characteristic of consequence-based calculi is a fine degree of control of interaction between consequences. For example, instead of keeping all derived consequences in a single set, the $\mathcal{ALCI}$ calculus by Simančík et al. (2014) constructs a graph-like structure called a *context structure*, where each node is called a *context* and is associated with a set of consequences. The calculus by Baader et al. (2005) also constructs a graph-like structure; and conjunctions on the left-hand sides of DL axioms correspond to contexts in the calculi by Kazakov (2009) and Simančík et al. (2011). In all of these cases, contexts encode an outline of a canonical ontology model, and they can be constructed in a goal-driven fashion reminiscent of hypertableau calculi. For example, a canonical model in our work is obtained by unfolding a context structure, so each element from the model's domain is obtained from the clauses associated with one context; we often informally say that a context *represents* the domain elements obtained from the context during such unfolding. A key benefit of contexts is that they 'localise' consequences to parts of a canonical model, which can be used to restrict inferences. For example, assume that an ontology contains a DL-clause $\alpha = B_1(x) \wedge B_2(x) \rightarrow B_3(x)$, and that consequences $\beta_1 = \top \rightarrow B_1(x)$ and $\beta_2 = \top \rightarrow B_2(x)$ have been derived. In standard resolution (Bachmair & Ganzinger, 2001), all consequences are kept in a single set, so whether $\gamma = \top \rightarrow B_3(x)$ is derived from α, β_1, and β_2 can be controlled only via the resolution parameters such as the atom ordering and the selection function. In contrast, resolution is possible in our calculus only if β_1 and β_2 belong to the same context, and γ is then derived only in that context. Thus, assigning consequences to contexts can restrict inferences much more than what is possible in standard resolution. After introducing our calculus formally, in Section 4.4.1 we show how this prevents the derivation of clauses (13) and (14) on the example ontology $\mathcal{O}_1$ from Figure 1.

The fourth characteristic of many consequence-based calculi is that they provide a degree of control in determining which contexts represent which elements from a canonical model. In the $\mathcal{ALCI}$ calculus by Simančík et al. (2014) and the work presented in this paper, each context is labelled by a conjunction of atoms called the *core*, which determines the atoms that hold for all elements represented by the context. A calculus parameter called the *expansion strategy* determines how cores are initialised, which can make contexts more or less specific: larger cores will typically make contexts represent more similar elements of a domain model (i.e., elements that participate in similar concepts and roles), which will usually lead to the derivation of fewer and/or simpler consequences spread across a larger number of contexts. We discuss these issues in more detail in Section 4.2. A similar level of control is possible in the calculus for $\mathcal{ALCH}$ (Simančík et al., 2011, Section 5).

Ontology $\mathcal{O}_2$

$B_0 \sqsubseteq \exists S^-.B_1$	$\rightsquigarrow$	$B_0(x) \rightarrow S(f_1(x), x)$	(16)
		$B_0(x) \rightarrow B_1(f_1(x))$	(17)
$B_1 \sqsubseteq \exists S.B_2$	$\rightsquigarrow$	$B_1(x) \rightarrow S(x, f_2(x))$	(18)
		$B_1(x) \rightarrow B_2(f_2(x))$	(19)
$B_1 \sqsubseteq \exists S.B_3$	$\rightsquigarrow$	$B_1(x) \rightarrow S(x, f_3(x))$	(20)
		$B_1(x) \rightarrow B_3(f_3(x))$	(21)
$B_2 \sqsubseteq B_4$	$\rightsquigarrow$	$B_2(x) \rightarrow B_4(x)$	(22)
$B_3 \sqsubseteq B_4$	$\rightsquigarrow$	$B_3(x) \rightarrow B_4(x)$	(23)
$B_2 \sqcap B_3 \sqsubseteq \bot$	$\rightsquigarrow$	$B_2(x) \wedge B_3(x) \rightarrow \bot$	(24)
$B_1 \sqsubseteq \leq 2.S$	$\rightsquigarrow$	$B_1(x) \wedge S(x, z_1) \wedge S(x, z_2) \wedge S(x, z_3) \rightarrow z_1 \approx z_2 \vee z_1 \approx z_3 \vee z_2 \approx z_3$	(25)

Figure 2: Example Ontology $\mathcal{O}_2$

3.3 Extending the Framework to $\mathcal{ALCHIQ}^+$

The main challenge in extending these ideas to $\mathcal{ALCHIQ}^+$ is in devising a representation of the consequences that can ensure completeness and termination. While existing calculi typically use DL-style axioms for that purpose, number restrictions and disjunctions can impose conditions that, we believe, cannot be represented using $\mathcal{ALCHIQ}^+$ axioms.

Let $\mathcal{O}_2$ be the ontology shown in Figure 2; please note that (16)–(17) are DL-clauses, even through the corresponding axiom in the DL notation contains an inverse role and is thus not normalised according to Table 1. In standard resolution, to prove $\mathcal{O}_2 \models B_0(x) \rightarrow B_4(x)$, we extend $\mathcal{O}_2$ with facts $B_0(a)$ and $\neg B_4(a)$ for a fresh constant a and apply the resolution inference rules. Then, clauses (16) and (17) derive $S(f_1(a), a)$ and $B_1(f_1(a))$; clauses (18) and (19) derive $S(f_1(a), f_2(f_1(a)))$ and $B_2(f_2(f_1(a)))$; and clauses (20) and (21) derive $S(f_1(a), f_3(f_1(a)))$ and $B_3(f_3(f_1(a)))$. Next, clause (22) derives $B_4(f_2(f_1(a)))$, and so clause (23) derives $B_4(f_3(f_1(a)))$. Moreover, clause (25) derives clause (26). Disjunct $f_3(f_1(a)) \approx f_2(f_1(a))$ of (26) cannot be satisfied since it is incompatible with the derived atoms $B_2(f_2(f_1(a)))$ and $B_3(f_3(f_1(a)))$ and clause (24). Thus, several further inferences derive (27), so $B_4(a)$ is derived regardless of which disjunct in the head of (27) holds.

$$f_2(f_1(a)) \approx a \vee f_3(f_1(a)) \approx a \vee f_3(f_1(a)) \approx f_2(f_1(a)) \tag{26}$$

$$f_2(f_1(a)) \approx a \vee f_3(f_1(a)) \approx a \tag{27}$$

Our calculus will need to represent (26) and (27), but this can be difficult to do using DL-style axioms: $\mathcal{ALCHIQ}^+$ axioms from Section 2.4 cannot talk about specific successors of a model element and so they cannot express (27), which essentially says 'either the second or the third successor of $f_1(a)$ are equal to a'. To address this issue, we represent consequences using *context clauses*—first-order clauses of a restricted shape where variable x represents domain elements represented by a context, and variable y and terms $f_i(x)$ provide names for the predecessor and the successors of the elements represented by x. We can thus capture (26) and (27) using context clauses (28) and (29), respectively.

$$f_2(x) \approx y \vee f_3(x) \approx y \vee f_3(x) \approx f_2(x) \tag{28}$$

$$f_2(x) \approx y \vee f_3(x) \approx y \tag{29}$$

As we discuss in Section 4, the rules of consequence-based calculi must be considerably modified to handle such clauses. In particular, context clauses contain equalities, which are not handled by the inference rules found in the consequence-based calculi known thus far. To handle equality, we have adapted the approaches of paramodulation calculi (Bachmair & Ganzinger, 1998) to the consequence-based framework. An important challenge was to ensure that our calculus mimics the inferences of the calculus by Baader et al. (2005) on $\mathcal{EL}$ ontologies. Thus, our calculus exhibits pay-as-you-go behaviour, and we intuitively believe that should ensure good performance on 'mostly-$\mathcal{EL}$' ontologies.

4. Formalising the Algorithm

We now formally define our consequence-based algorithm for $\mathcal{ALCHIQ}^+$, discuss its formal properties, and complete the presentation of the examples from Section 3.

4.1 Definitions

As we have explained in Section 3.2, a key aspect of our calculus is that consequences are not kept in a single set, but are associated with *contexts*—that is, vertices of a directed labelled graph called a *context structure* that summarises a canonical model in a particular way. Consequences assigned to a context are called *context clauses* and are constructed from *context terms* and *context literals* as described in Definition 1. We restrict context clauses to contain only variables x and y, which have a special meaning in our setting: variable x represents a ground term t from a canonical Herbrand model of an ontology, and y represents the predecessor of t; this naming convention is important for the inference rules of our calculus. This is different to the DL-clauses of an ontology, which can contain variables x and z_i, and where z_i refer to either the predecessor or a successor of x.

Unless stated otherwise, in the rest of this section B is an atomic concept of type $\mathsf{a} \to \mathsf{p}$, S is an atomic role of type $\mathsf{a} \times \mathsf{a} \to \mathsf{p}$, and f is a successor function of type $\mathsf{a} \to \mathsf{a}$. Moreover, please remember that true is a special constant used in the encoding of predicate symbols as function symbols, and that we often abbreviate atoms $A \approx \mathsf{true}$ as just A (see Section 2.2).

Definition 1. *A* context a-term *is a term of the form y, x, or $f(x)$; a* context p-term *is a term of the form $B(y)$, $B(x)$, $B(f(x))$, $S(x, y)$, $S(y, x)$, $S(x, x)$, $S(x, f(x))$, $S(f(x), x)$, or the constant true; and a* context term *is a context a-term or a context p-term. A* context atom *is an equality of the form $A \approx \mathsf{true}$ where A is a context p-term other than true; and a* context literal *is a context atom or a literal of the form $l \bowtie r$ where l and r are context a-terms and $\bowtie \in \{\approx, \not\approx\}$. A* context clause *contains only context atoms of the form $B(x)$, $S(x, x)$, $S(y, x)$, and $S(x, y)$ in the body and only context literals in the head.*

For a given ontology $\mathcal{O}$, sets $\mathsf{Su}(\mathcal{O})$ and $\mathsf{Pr}(\mathcal{O})$ from Definition 2 identify the information that must be exchanged between adjacent contexts in a context structure. We explain the intuition behind these sets after formally introducing the notion of context structure.

Definition 2. *The set $\mathsf{Su}(\mathcal{O})$ of* successor triggers *of an ontology $\mathcal{O}$ is the smallest set of atoms such that, for each DL-clause $\Gamma \to \Delta \in \mathcal{O}$,*

- $B(x) \in \Gamma$ *implies* $B(x) \in \mathsf{Su}(\mathcal{O})$,
- $S(x, z_i) \in \Gamma$ *implies* $S(x, y) \in \mathsf{Su}(\mathcal{O})$, *and*
- $S(z_i, x) \in \Gamma$ *implies* $S(y, x) \in \mathsf{Su}(\mathcal{O})$.

The set $\mathsf{Pr}(\mathcal{O})$ *of* predecessor triggers *of* $\mathcal{O}$ *is defined as the set of literals*

$$\mathsf{Pr}(\mathcal{O}) = \{\, A\{x \mapsto y,\ y \mapsto x\} \mid A \in \mathsf{Su}(\mathcal{O}) \,\} \cup \{\, B(y) \mid B \text{ occurs in } \mathcal{O} \,\} \cup \{x \approx y\}.$$

As in standard resolution (Bachmair & Ganzinger, 2001), we order clause literals using a term order $\succ$ and allow only maximal literals to participate in inferences. Definition 3 specifies the conditions that such $\succ$ must satisfy. The term order will need not be the same for all contexts of a context structure. In particular, a-terms will have to be compared in the same way in all contexts; we ensure this by basing $\succ$ on a total oder $\succcurlyeq$ on all function symbols of sort a, and $\succcurlyeq$ will be global for all contexts. In contrast, for reasons we discuss in Section 4.3, we will allow p-terms to be compared differently in different contexts. Conditions 1 through 4 ensure that, if we ground the order by mapping x to a term t and y to the predecessor of t, we obtain a *simplification order* (Baader & Nipkow, 1998). Finally, condition 5 ensures that atoms that might be propagated to a predecessor of a context are smallest in the ordering; we discuss in Section 4.4 why this is important for completeness.

Definition 3. *Let* $\succcurlyeq$ *be a total, well-founded order on function symbols of sort* a. *A* context term order $\succ$ *w.r.t.* $\succcurlyeq$ *is a strict order on context terms satisfying the following conditions:*

1. *for each context* p*-term* A *with* $A \neq \mathsf{true}$, *we have* $A \succ x \succ y \succ \mathsf{true}$;
2. *for all* $f, g \in \Sigma^F_{\mathsf{a}}$ *with* $f \succcurlyeq g$, *we have* $f(x) \succ g(x)$;
3. *for all context terms* s_1 *and* s_2 *such that* $s_2 \succ s_1$, *we have* $t[s_2]_p \succ t[s_1]_p$ *for each context term* t *and each position* p *in* t;
4. *for each context term* s *and each proper position* p *in* s, *we have* $s \succ s|_p$; *and*
5. *for each atom* $A \approx \mathsf{true} \in \mathsf{Pr}(\mathcal{O})$ *and each context term* $s \notin \{x, y, \mathsf{true}\}$, *we have* $A \not\succ s$.

Each term order is extended to a literal order, also written $\succ$, *as described in Section 2.3.*

We can obtain a context term order $\succ$ as follows. We fix a total, well-founded order $\succcurlyeq$ on the symbols of sort a, and we extend it (arbitrarily) to also compare all symbols of sort p, and variables x and y. Next, we let $\succ$ be the *lexicographic path order* (LPO) (Baader & Nipkow, 1998) over context a- and p-terms induced by $\succcurlyeq$, where we treat variables x and y as constants of sort a. Order $\succcurlyeq$ is total on all symbols occurring in context terms, so $x \succcurlyeq y$ and the well-known properties of LPOs ensure that $\succ$ is a total simplification order on all context terms that satisfies conditions 1 through 4 of Definition 3. To also satisfy condition 5, we relax $\succ$ by dropping all $A \succ s$ where $A \approx \mathsf{true} \in \mathsf{Pr}(\mathcal{O})$ and $s \notin \{x, y, \mathsf{true}\}$. Condition 5 is clearly satisfied after this step; conditions 1 and 2 remain satisfied because the relaxation step does not change the order between p-terms and x, y, and true, or between functional a-terms; and conditions 3 and 4 remain satisfied because no new pairs are added

to the order and the eliminated pairs are not of the form $t[s_2]_p \succ t[s_1]_p$ or $s \succ s|_p$ for some terms t, s, s_1, s_2, and nonempty position p.

Effective redundancy elimination techniques are critical to efficiency of resolution calculi, and Definition 4 defines a notion compatible with our framework.

Definition 4. *A set of clauses U* contains a clause $\Gamma \to \Delta$ up to redundancy*, written* $\Gamma \to \Delta \mathrel{\hat{\in}} U$*, if*

1. $s \approx s \in \Delta$ *or* $\{s \approx s',\ s \not\approx s'\} \subseteq \Delta$ *for some terms s and s', or*
2. $\Gamma' \subseteq \Gamma$ *and* $\Delta' \subseteq \Delta$ *for some clause* $\Gamma' \to \Delta' \in U$.

If U contains $\Gamma \to \Delta$ up to redundancy, then adding $\Gamma \to \Delta$ to U does not modify the constraints that U represents because either $\Gamma \to \Delta$ is a tautology (the first case) or U contains a stronger clause (the second case). Note that clauses $A \to A$ are *not* redundant: they specify that atom A may hold in a context. Moreover, the inference rules that we present shortly will ensure that, whenever a clause of the form $\Gamma \wedge A \to \Delta \vee A$ is derived in a context v, then v will have been initialised with $\top \to A$ or $A \to A$, and so $\Gamma \wedge A \to \Delta \vee A$ will be redundant by condition 2 of Definition 4. The usual tautology elimination rules apply to clause heads; hence, clauses $\Gamma \to \Delta \vee s \approx s$ and $\Gamma \to \Delta \vee s \approx s' \vee s \not\approx s'$ can both be eliminated. Proposition 5 shows that we can remove from U each clause C that is contained in $U \setminus \{C\}$ up to redundancy, which our calculus uses to support backward redundancy elimination via the Elim rule (see Table 2).

Proposition 5. *Given a set of clauses U and clauses C and C' such that $C \mathrel{\hat{\in}} U \setminus \{C\}$ and $C' \mathrel{\hat{\in}} U$, we have $C' \mathrel{\hat{\in}} U \setminus \{C\}$.*

Definition 6 formalises the notion of a context structure as a directed graph whose edges are labelled by function symbols. Intuitively, each context (i.e., a graph vertex) v represents one or more terms from a canonical Herbrand model of an ontology, and it is associated with a set $\mathcal{S}_v$ of context clauses, a conjunction of atoms core_v, and a term order $\succ_v$ used to restrict the inferences involving the clauses of $\mathcal{S}_v$. To ensure that a-terms are ordered in the same way in all contexts, each term order $\succ_v$ is defined w.r.t. a global order $\succcurlyeq$ on function symbols of sort a. Conjunction core_v determines the 'type' of context v—that is, it specifies atoms that necessarily hold for all terms in a canonical model that are represented by v, and so it indirectly determines which terms of a canonical model are represented by v. We discuss this shortly after introducing the Succ inference rule, as well as in Section 4.2.

Definition 6 also specifies conditions that ensure that a context structure only represents the consequences of an ontology. Since core_v holds implicitly for each term in a model of $\mathcal{O}$ represented by context v, conjunction core_v is not included in the bodies of the clauses in $\mathcal{S}_v$. Then, condition S1 of Definition 6 says that each clause in $\mathcal{S}_v$ extended with core_v in the body must be a consequence of $\mathcal{O}$; please note that Γ and/or Δ can be empty. Moreover, condition S2 ensures that each context clause derived by the Pred rule of our calculus (to be defined shortly) is indeed a consequence of $\mathcal{O}$.

Definition 6. *A* context structure *for an ontology $\mathcal{O}$ is a tuple $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathsf{core}, \succcurlyeq, \succ \rangle$, where $\mathcal{V}$ is a finite set of* contexts*; $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \Sigma_{\mathsf{a}}^F$ is a finite set of edges each of which is labelled by a function symbol; function* core *assigns to each context $v \in \mathcal{V}$ a conjunction*

core_v *of atoms of the form* $B(x)$, $S(x,x)$, $S(x,y)$, *or* $S(y,x)$; *function* $\mathcal{S}$ *assigns to each context* $v \in \mathcal{V}$ *a finite set* $\mathcal{S}_v$ *of context clauses;* $\succcurlyeq$ *is a total, well-founded order on function symbols of sort* a; *and function* $\succ$ *assigns to each context* $v \in \mathcal{V}$ *a context term order* $\succ_v$ *w.r.t.* $\succcurlyeq$. *A context structure* $\mathcal{D}$ *is* sound *for* $\mathcal{O}$ *if the following conditions both hold.*

S1. For each context $v \in \mathcal{V}$ *and each clause* $\Gamma \to \Delta \in \mathcal{S}_v$, *we have* $\mathcal{O} \models \mathsf{core}_v \wedge \Gamma \to \Delta$.

S2. For each edge $\langle u, v, f \rangle \in \mathcal{E}$, *we have* $\mathcal{O} \models \mathsf{core}_u \to \mathsf{core}_v\{x \mapsto f(x), y \mapsto x\}$.

There is considerable freedom in determining what terms are represented by a context. We can make contexts more specific (i.e., make them represent fewer terms); this will usually lead to the derivation of fewer consequences in each context, but the number of contexts might increase. We can also make contexts less specific; this will usually reduce the number of contexts, but the number of consequences in each context might increase. Simančík et al. (2014) show that there is an inherent tension between these two tendencies, and they discuss how this affects the computational complexity of reasoning. We next introduce the notion of an expansion strategy—a parameter of our calculus that determines when and how to create/reuse contexts. We present several concrete and practically relevant strategies in Section 4.2. We discuss the intuition behind this definition shortly.

Definition 7. *An* expansion strategy *is a polynomial-time function* $\mathsf{strategy}$ *that takes as arguments a function symbol* f, *a finite set of atoms* $K \subseteq \mathsf{Su}(\mathcal{O})$, *and a context structure* $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathsf{core}, \succcurlyeq, \succ \rangle$ *for* $\mathcal{O}$. *The result of* $\mathsf{strategy}(f, K, \mathcal{D})$ *is a triple* $\langle v, \mathsf{core}', \succ' \rangle$ *where*

- core' *is a subset of* K;
- *either* $v \notin \mathcal{V}$ *is a fresh context not occurring in* $\mathcal{D}$, *or* $v \in \mathcal{V}$ *is an existing context from* $\mathcal{D}$ *such that* $\mathsf{core}_v = \mathsf{core}'$ *and* $\succ' \subseteq \succ$; *and*
- $\succ'$ *is a context term order w.r.t.* $\succcurlyeq$.

Table 2 shows the inference rules of our calculus. A rule can introduce an edge, introduce and initialise a context, derive a clause, or delete a clause. Inferences are restricted to the clause literals that are maximal under the relevant context term order. We next discuss the high-level intuition behind different rules. We discuss various issues in more detail Section 4.4, where we apply the calculus to ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ from Section 3.

The Core rule ensures that all atoms from a context's core indeed hold for the terms represented by the context.

The Hyper rule implements hyperresolution between the DL-clauses of the ontology and the clauses of one context. The main difference to standard hyperresolution is that x must match to x, the importance of which we discuss in detail in Section 4.4.

The Eq, Ineq, and Factor rules implement equality reasoning by adapting analogous rules from first-order equational calculi (Bachmair & Ganzinger, 1998).

The Elim rule eliminates redundant clauses, which is possible due to Proposition 5. This rule is not strictly needed for completeness, but it plays a critical role in practice. Thus, for the sake of completeness, we present this rule together with the other rules.

The Succ rule extends the context structure so that each context has the relevant successor contexts. Set $\mathsf{Su}(\mathcal{O})$ of successor triggers determines which information must be

Table 2: Rules of the Consequence-Based Calculus

Core	If	$A \in \mathsf{core}_v$,
		and $\top \to A \,\hat{\notin}\, \mathcal{S}_v$,
	then	add $\top \to A$ to $\mathcal{S}_v$.
Hyper	If	$\bigwedge_{i=1}^{n} A_i \to \Delta \in \mathcal{O}$,
		σ is a substitution such that $\sigma(x) = x$,
		$\Gamma_i \to \Delta_i \vee A_i\sigma \in \mathcal{S}_v$ with $\Delta_i \not\succeq_v A_i\sigma$ for $1 \leq i \leq n$,
		and $\bigwedge_{i=1}^{n} \Gamma_i \to \Delta\sigma \vee \bigvee_{i=1}^{n} \Delta_i \,\hat{\notin}\, \mathcal{S}_v$,
	then	add $\bigwedge_{i=1}^{n} \Gamma_i \to \Delta\sigma \vee \bigvee_{i=1}^{n} \Delta_i$ to $\mathcal{S}_v$.
Eq	If	$\Gamma_1 \to \Delta_1 \vee s_1 \approx t_1 \in \mathcal{S}_v$ with $s_1 \succ_v t_1$ and $\Delta_1 \not\succeq_v s_1 \approx t_1$,
		$\Gamma_2 \to \Delta_2 \vee s_2 \bowtie t_2 \in \mathcal{S}_v$ with $\bowtie \in \{\approx, \not\approx\}$ and $s_2 \succ_v t_2$ and $\Delta_2 \not\succeq_v s_2 \bowtie t_2$,
		p is a position such that $s_2|_p = s_1$ and $s_2|_p$ is not a variable,
		and $\Gamma_1 \wedge \Gamma_2 \to \Delta_1 \vee \Delta_2 \vee s_2[t_1]_p \bowtie t_2 \,\hat{\notin}\, \mathcal{S}_v$,
	then	add $\Gamma_1 \wedge \Gamma_2 \to \Delta_1 \vee \Delta_2 \vee s_2[t_1]_p \bowtie t_2$ to $\mathcal{S}_v$.
Ineq	If	$\Gamma \to \Delta \vee t \not\approx t \in \mathcal{S}_v$,
		and $\Gamma \to \Delta \,\hat{\notin}\, \mathcal{S}_v$,
	then	add $\Gamma \to \Delta$ to $\mathcal{S}_v$.
Factor	If	$\Gamma \to \Delta \vee s \approx t \vee s \approx t' \in \mathcal{S}_v$ with $\Delta \cup \{s \approx t\} \not\succeq_v s \approx t'$ and $s \succ_v t'$,
		and $\Gamma \to \Delta \vee t \not\approx t' \vee s \approx t' \,\hat{\notin}\, \mathcal{S}_v$,
	then	add $\Gamma \to \Delta \vee t \not\approx t' \vee s \approx t'$ to $\mathcal{S}_v$.
Elim	If	$\Gamma \to \Delta \in \mathcal{S}_v$ and
		$\Gamma \to \Delta \,\hat{\in}\, \mathcal{S}_v \setminus \{\Gamma \to \Delta\}$,
	then	remove $\Gamma \to \Delta$ from $\mathcal{S}_v$.
Pred	If	$\langle u, v, f\rangle \in \mathcal{E}$,
		$\bigwedge_{i=1}^{m} A_i \to \bigvee_{i=m+1}^{m+n} L_i \in \mathcal{S}_v$,
		$\Gamma_i \to \Delta_i \vee A_i\sigma \in \mathcal{S}_u$ with $\Delta_i \not\succeq_u A_i\sigma$ for $1 \leq i \leq m$,
		$L_i \in \mathsf{Pr}(\mathcal{O})$ for each $m+1 \leq i \leq m+n$,
		and $\bigwedge_{i=1}^{m} \Gamma_i \to \bigvee_{i=1}^{m} \Delta_i \vee \bigvee_{i=m+1}^{m+n} L_i\sigma \,\hat{\notin}\, \mathcal{S}_u$,
	then	add $\bigwedge_{i=1}^{m} \Gamma_i \to \bigvee_{i=1}^{m} \Delta_i \vee \bigvee_{i=m+1}^{m+n} L_i\sigma$ to $\mathcal{S}_u$;
	where	$\sigma = \{x \mapsto f(x), y \mapsto x\}$.
Succ	If	$\Gamma \to \Delta \vee A \in \mathcal{S}_u$ where $\Delta \not\succeq_u A$ and A contains $f(x)$, and
		no edge $\langle u, v, f\rangle \in \mathcal{E}$ exists such that $A' \to A' \,\hat{\in}\, \mathcal{S}_v$ for each $A' \in K_2 \setminus \mathsf{core}_v$,
	then	let $\langle v, \mathsf{core}', \succ'\rangle := \mathsf{strategy}(f, K_1, \mathcal{D})$;
		if $v \notin \mathcal{V}$, then add v to $\mathcal{V}$ and let $\mathsf{core}_v := \mathsf{core}'$, $\succ_v := \succ'$, and $\mathcal{S}_v := \emptyset$;
		if $\langle u, v, f\rangle \notin \mathcal{E}$, then add $\langle u, v, f\rangle$ to $\mathcal{E}$; and
		add $A' \to A'$ to $\mathcal{S}_v$ for each $A' \in K_2 \setminus \mathsf{core}_v$;
	where	$\sigma = \{x \mapsto f(x), y \mapsto x\}$,
		$K_1 = \{\, A' \in \mathsf{Su}(\mathcal{O}) \mid \top \to A'\sigma \in \mathcal{S}_u \,\}$, and
		$K_2 = \{\, A' \in \mathsf{Su}(\mathcal{O}) \mid \Gamma' \to \Delta' \vee A'\sigma \in \mathcal{S}_u$ and $\Delta' \not\succeq_u A'\sigma \,\}$.

propagated to such contexts. Assume that a context structure contains an f-labelled edge from context u to context v; hence, if u represents a term t in a model I of $\mathcal{O}$, then v represents the term $f(t)$. Now assume that context u contains a context clause containing $B(f(x))$ in its head, and that a DL-clause of an ontology contains a body atom $B(x)$. The context clause in u can require $B(f(t))$ to hold in I; however, the Hyper rule must be applied to the DL-clause in context v because that context is responsible for describing the conditions on the part of I that involves $f(t)$, its predecessor t, and the successors of $f(t)$. Thus, while atom $B(f(t))$ from I is represented as $B(f(x))$ in context u, the atom must also be reflected in context v as $B(x)$ to allow correct application of the Hyper rule. To that end, $\mathsf{Su}(\mathcal{O})$ contains $B(x)$ as a signal that $B(f(x))$ in context u should be propagated as $B(x)$ to context v. Set $\mathsf{Su}(\mathcal{O})$ contains only atoms that can participate in an inference in v and are thus relevant for v, which prevents unnecessary propagation of information from u to v. Based on these observations, the Succ rule is applicable in context u if a function symbol f occurs in a maximal head literal of a context clause, but there is no appropriate f-labelled edge from u: set K_1 identifies the atoms of $\mathsf{Su}(\mathcal{O})$ that are known to hold in the successor context, and set K_2 identifies the atoms of $\mathsf{Su}(\mathcal{O})$ that can, but are not certain to hold in the successor context; note that $K_1 \subseteq K_2$ always holds. The rule consults the expansion strategy, which can reuse an existing context or introduce a new one; in the latter case, the strategy also designates a subset of K_1 as the core of the new context and determines its term order. Based on the strategy's output, the Succ rule extends the context structure by adding a context and/or edge as needed and initialising the context with the relevant atoms from $K_2 \setminus \mathsf{core}_v$; atoms in core_v are excluded as they are initialised by the Core rule.

The Pred rule can be understood as hyperresolution between a context v and a predecessor context u. Clause $C = \bigwedge_{i=1}^{m} A_i \rightarrow \bigvee_{i=m+1}^{m+n} L_i$ plays the role of the main premise, and from the perspective of context u it 'looks like' $C\sigma = \bigwedge_{i=1}^{m} A_i\sigma \rightarrow \bigvee_{i=m+1}^{m+n} L_i\sigma$ for σ as specified in the inference. Set $\mathsf{Pr}(\mathcal{O})$ of predecessor triggers contains literals that are of interest to u. Atom $B(y)$ in context v is represented as atom $B(x)$ in u, so $\mathsf{Pr}(\mathcal{O})$ contains $B(y)$ to indicate that consequences containing $B(y)$ in context v should be propagated to context u as $B(x)$. Note that all elements of $\mathsf{Pr}(\mathcal{O})$ are of the form $B(y) \approx \mathsf{true}$, $S(x, y) \approx \mathsf{true}$, $S(y, x) \approx \mathsf{true}$, or $x \approx y$. Note also that $\mathsf{Su}(\mathcal{O})$ does not contain $x \approx y$: such atoms need not be propagated to a successor context since they cannot be matched to a DL-clause body. Based on these observations, $L_i \in \mathsf{Pr}(\mathcal{O})$ for $m + 1 \leq i \leq m + n$ ensures that all atoms from the head of $C\sigma$ are relevant to and can be captured in context u. Thus, the Pred rule hyperresolves each atom $A_i\sigma$ by a clause $\Gamma_i \rightarrow \Delta_i \vee A_i\sigma$ from context u.

4.2 Expansion Strategies

Simančík et al. (2014) presented three natural and practically relevant strategies. We can adapt their discussion to the $\mathcal{ALCHIQ}^+$ setting as follows.

- The *trivial* strategy introduces just one context $v_\top$ with the empty core—that is, $\mathsf{core}_{v_\top} = \top$. All consequences then belong to a single set. Nevertheless, the calculus is still different to the known resolution-based procedures: the Hyper, Succ, and Pred ensure completeness without deriving terms of unbounded depth.

- The *eager* strategy returns for each K_1 the context v_{K_1} with core K_1. Then v_{K_1} represents fewer ground terms, which is likely to reduce the number of clauses associated with v_{K_1}; however, the number of contexts can be exponential in the size of $\mathcal{O}$.
- The *cautious* strategy examines the function symbol f: if f occurs in $\mathcal{O}$ in exactly one atom of the form $B(f(x))$ and if $B(x) \in K_1$, then the result is the context v_B with core $B(x)$; otherwise, the result is the 'trivial' context $v_\top$ with the empty core. Contexts are then less constrained than with the eager strategy, but the number of contexts is at most linear in the size of $\mathcal{O}$. This strategy captures how contexts are created in consequence-based calculi for $\mathcal{EL}$, where all existential restrictions of the form $\exists S.B$ are satisfied in a single context v_B.

Simančík et al. (2014) discuss the relative merits of these strategies; even though their discussion is based on $\mathcal{ALCI}$, their conclusions apply to our setting as well. They show that the eager strategy is generally most effective at reducing unnecessary inferences: it produces more fine-grained contexts, which generally leads to the derivation of fewer and shorter clauses. On some ontologies, however, the eager strategy can produce a large number of contexts, so the cautious strategy is more appropriated when memory is limited. Finally, a reasoner can start with the eager strategy, but switch to the cautious one if it starts running out of memory. Both strategies are almost always superior to the trivial one.

4.3 Consequence-Based Algorithm and Its Properties

We can obtain a sound and complete DL reasoning algorithm by applying the inference rules from Table 2 to an appropriately initialised context structure. Towards this goal, Theorem 8 captures two important properties: applying an inference rule from Table 2 to a context structure produces a context structure (i.e., the result satisfies Definition 6), and the derived clauses are indeed entailed by the ontology (but please remember that each consequence is relative to the core of its context). The theorem is proved in Appendix A.

Theorem 8 (Soundness)**.** *For an arbitrary expansion strategy, applying an inference rule from Table 2 to an ontology $\mathcal{O}$ and a context structure $\mathcal{D}$ that is sound for $\mathcal{O}$ produces a context structure that is sound for $\mathcal{O}$.*

Theorem 9 shows that our inference rules are complete in the following sense. Assume that we wish to check $\mathcal{O} \models \Gamma_Q \to \Delta_Q$ for $\Gamma_Q \to \Delta_Q$ a query clause—that is, Γ_Q and Δ_Q consist of atoms of the form $B(x)$ and $S(x, x)$. If a context structure $\mathcal{D}$ is saturated under the inference rules from Table 2, and if $\mathcal{D}$ contains a context q that is initialised according to conditions C2 and C3 of Theorem 9, then $\mathcal{O} \models \Gamma_Q \to \Delta_Q$ implies $\Gamma_Q \to \Delta_Q \,\hat{\in}\, \mathcal{S}_q$—that is, unless $\Gamma_Q \to \Delta_Q$ is a tautology according to condition 1 of Definition 4, a stronger clause $\Gamma' \to \Delta'$ with $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$ is derived in context q. Condition C2 requires all atoms of Δ_Q to be minimal in the term order $\succ_q$ of context q (apart from the atoms that contain variable y, which are necessarily smaller). This is analogous to the *answer literal* technique (Green, 1969) from first-order theorem proving, which can be used to derive all answers to a query in ordered resolution. Condition C2 applies only to context q; thus, since Definition 6 allows the term order to vary across contexts, we can use a stronger order in contexts other than q and thus possibly restrict inferences. Condition C3 requires q to be initialised with

each atom from Γ_Q. This is analogous to how a (hyper)tableau calculus would be initialised to check the satisfiability of Γ_Q. The theorem is proved in Appendix C.

Theorem 9 (Completeness)**.** *Let $\mathcal{O}$ be an ontology, and let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathsf{core}, \succcurlyeq, \succ \rangle$ be a context structure such that no inference rule from Table 2 is applicable to $\mathcal{O}$ and $\mathcal{D}$. Then, for each query clause $\Gamma_Q \to \Delta_Q$ and each context $q \in \mathcal{V}$ such that conditions C1 through C3 are satisfied, $\Gamma_Q \to \Delta_Q \mathrel{\hat{\in}} \mathcal{S}_q$ holds.*

C1. $\mathcal{O} \models \Gamma_Q \to \Delta_Q$.

C2. For each context atom $A \approx \mathsf{true} \in \Delta_Q$ and each context term t such that t does not contain variable y and $A \succ_q t$ holds, we have $t \approx \mathsf{true} \in \Delta_Q$.

C3. For each context atom $A \approx \mathsf{true} \in \Gamma_Q$, we have $\Gamma_Q \to A \approx \mathsf{true} \mathrel{\hat{\in}} \mathcal{S}_q$.

Theorem 9 just requires $\mathcal{D}$ to be saturated, and the preconditions of the Succ rule do not mention an expansion strategy; thus, our claim is independent from the expansion strategy that is used to saturate $\mathcal{D}$. This is analogous to the formal development of resolution (Bachmair & Ganzinger, 2001): any saturated set of clauses that does not contain the empty clause is satisfiable, regardless of the strategy used to construct this set.

Algorithm 1 provides us with a concrete procedure for checking entailment of a query clause. The algorithm selects an expansion strategy and initialises $\mathcal{D}$ to an empty context structure. The algorithm then introduces a context q into the context structure whose core is initialised to Γ_Q (so the Core rule eventually ensures condition C3 of Theorem 9), and whose order $\succ_q$ satisfies condition C2 of Theorem 9 (which can be obtained by relaxing an order as described after Definition 3). The algorithm then saturates $\mathcal{D}$ using the rules from Table 2, after which Theorems 8 and 9 guarantee $\Gamma_Q \to \Delta_Q \mathrel{\hat{\in}} \mathcal{S}_q$ if and only if $\mathcal{O} \models \Gamma_Q \to \Delta_Q$.

This algorithm can be adapted to decide entailment of several query clauses in one run. First, one can repeat line A2 for each target query clause independently before proceeding to the saturation step. This can be impractical when the number of query clauses is large (e.g., when classifying an ontology), in which case one can adapt line A2 and satisfy conditions C2 and C3 of Theorem 9 for multiple query clauses using just one context. For example, to decide entailment of $B(x) \to B_i(x)$ for $1 \leq i \leq n$, we can introduce one context q with $\mathsf{core}_q = B(x)$ and initialise $\succ_q$ so that all B_i are incomparable; then, for each i, $B(x) \to B_i(x) \mathrel{\hat{\in}} \mathcal{S}_q$ holds upon algorithm's termination if and only if $\mathcal{O} \models B(x) \to B_i(x)$.

Algorithm 1 is also well-suited to deciding entailment of query clauses that are generated incrementally. For example, the algorithm by Glimm, Horrocks, Motik, Shearer, and Stoilos (2012) can classify an ontology by iteratively checking entailment of query clauses while trying to minimise the number of such checks. In such a case, it may be helpful to omit line A1 before each run of Algorithm 1 and incrementally expand the context structure, thus implicitly reusing in each run the work from all previous runs.

Algorithm 1 may not terminate if a strategy always chooses to introduce a fresh context (which is possible by Definition 7). By Proposition 10, however, the algorithm terminates whenever a strategy introduces finitely many contexts, and it becomes worst-case optimal for $\mathcal{ALCHIQ}^+$ if the number of contexts is at most exponential. Since our algorithm takes as input a set of DL-clauses $\mathcal{O}$, we state this complexity result w.r.t. the size of $\mathcal{O}$. If an ontology is written using the DL notation from Section 2.4, these results hold if the numbers

Algorithm 1 Deciding $\mathcal{O} \models \Gamma_Q \to \Delta_Q$

A1. Create an empty context structure $\mathcal{D}$ and select an expansion strategy.

A2. Introduce a context q into $\mathcal{D}$, set $\mathsf{core}_q = \Gamma_Q$, and initialise the order $\succ_q$ in a way that satisfies condition C2 of Theorem 9.

A3. Apply the inference rules from Table 2 to $\mathcal{D}$ and $\mathcal{O}$ until no inference rule is applicable.

A4. $\mathcal{O} \models \Gamma_Q \to \Delta_Q$ holds if and only if $\Gamma_Q \to \Delta_Q \,\hat{\in}\, \mathcal{S}_q$.

in concepts $\geqslant n\, R.C$ and $\leqslant n\, R.C$ are coded in unary; otherwise, the transformation into DL-clauses incurs an exponential blowup. Most decision procedures for DLs based on resolution or (hyper)tableau follow this assumption.

Proposition 10. *Algorithm 1 terminates whenever the expansion strategy introduces finitely many contexts in the algorithm's run. The algorithm runs in worst-case exponential time in the size of $\mathcal{O}$ if the number of introduced contexts is at most exponential in the size of $\mathcal{O}$.*

Proposition 11 shows that our algorithm is worst-case optimal on $\mathcal{ELH}$ ontologies; moreover, Proposition 12 shows that, on $\mathcal{EL}$ ontologies, our calculus with the cautious strategy simulates the inferences by Baader et al. (2005).

Proposition 11. *On $\mathcal{ELH}$ ontologies and queries of the form $B_1(x) \to B_2(x)$, Algorithm 1 runs in polynomial time in the size of $\mathcal{O}$ with either the cautious or the eager strategy.*

Proposition 12. *If the ontology is in $\mathcal{EL}$, the context structure is initialised by a context v_B with $\mathsf{core}_{v_B} = \{B(x)\}$ for each atomic concept B, the* Hyper *rule is applied eagerly, and the cautious strategy is used, then numbers of inferences in step A3 of Algorithm 1 and of the calculus by Baader et al. (2005) are in a linear relationship.*

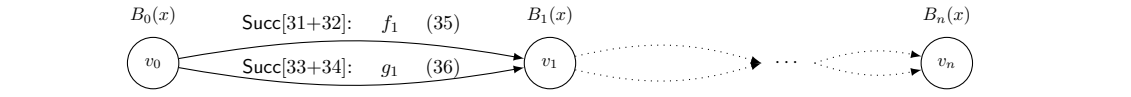
The correspondence of inferences in Proposition 12 is due to the definition of $\mathsf{Pr}(\mathcal{O})$ and the shape of $\mathcal{EL}$ DL-clauses from Table 1. In particular, if $\mathcal{O}$ is an $\mathcal{EL}$ ontology, then each binary atom in $\mathcal{O}$ is of the form $S(z_1, x)$ or $S(x, f(x))$, so binary atoms in $\mathsf{Su}(\mathcal{O})$ are of the form $S(y, x)$. Moreover, due to the absence of role hierarchies, whenever a clause of the form $S(y, x) \to S(y, x)$ or $S(y, x) \to B(y)$ is derived in a context v, there exists a predecessor context v' of v that contains $\top \to S(x, f(x))$, and these are all possible forms of context clauses with binary atoms. Therefore, the derivation of a context clause $S(y, x) \to B(y)$ is always followed by an application the Pred rule, which corresponds to an application of the rule CR4 by Baader et al. (2005). In fact, on $\mathcal{EL}$, our calculus is closest to the calculus by Kazakov, Krötzsch, and Simančík (2011), which materialises auxiliary axioms $\exists S.B' \sqsubseteq \exists S.B$ that correspond to context clauses $S(y, x) \to B(y)$ in our calculus.

4.4 Examples

With the calculus formally defined, we now complete our discussion from Sections 3.1 and 3.3 and show how our calculus handles the two examples mentioned there.

Ontology $\mathcal{O}_1$

$B_i \sqsubseteq \exists S_1.B_{i+1}$	$\rightsquigarrow$	$B_i(x) \rightarrow S_1(x, f_{i+1}(x))$	(4)	for $0 \leq i < n$
		$B_i(x) \rightarrow B_{i+1}(f_{i+1}(x))$	(5)	
$B_i \sqsubseteq \exists S_2.B_{i+1}$	$\rightsquigarrow$	$B_i(x) \rightarrow S_2(x, g_{i+1}(x))$	(6)	
		$B_i(x) \rightarrow B_{i+1}(g_{i+1}(x))$	(7)	
$B_n \sqsubseteq C_n$	$\rightsquigarrow$	$B_n(x) \rightarrow C_n(x)$	(8)	
$\exists S_1.C_{i+1} \sqsubseteq C_i$	$\rightsquigarrow$	$S_1(z_1, x) \wedge C_{i+1}(x) \rightarrow C_i(z_1)$	(9)	for $0 \leq i < n$
$\exists S_2.C_{i+1} \sqsubseteq C_i$	$\rightsquigarrow$	$S_2(z_1, x) \wedge C_{i+1}(x) \rightarrow C_i(z_1)$	(10)	
$C_0 \sqcap \cdots \sqcap C_n \sqsubseteq \bot$	$\rightsquigarrow$	$C_0(x) \wedge \cdots \wedge C_n(x) \rightarrow \bot$	(11)	



Core:	$\top \rightarrow B_0(x)$	(30)	Succ[31+32]:	$S_1(y,x) \rightarrow S_1(y,x)$	(37)	Succ[...]:	$S_1(y,x) \rightarrow S_1(y,x)$ (44)
Hyper[4+30]:	$\top \rightarrow S_1(x, f_1(x))$	(31)	Succ[33+34]:	$S_2(y,x) \rightarrow S_2(y,x)$	(38)	Succ[...]:	$S_2(y,x) \rightarrow S_2(y,x)$ (45)
Hyper[5+30]:	$\top \rightarrow B_1(f_1(x))$	(32)	Core:	$\top \rightarrow B_1(x)$	(39)	Core:	$\top \rightarrow B_n(x)$ (46)
Hyper[6+30]:	$\top \rightarrow S_2(x, g_1(x))$	(33)	Hyper[4+39]:	$\top \rightarrow S_1(x, f_2(x))$	(40)	Hyper[8+46]:	$\top \rightarrow C_n(x)$ (47)
Hyper[7+30]:	$\top \rightarrow B_1(g_1(x))$	(34)	Hyper[5+39]:	$\top \rightarrow B_2(f_2(x))$	(41)	Hyper[9+44+47]:	$S_1(y,x) \rightarrow C_{n-1}(y)$ (48)
Pred[51]:	$\top \rightarrow C_0(x)$	(53)	Hyper[6+39]:	$\top \rightarrow S_2(x, g_2(x))$	(42)	Hyper[10+45+47]:	$S_2(y,x) \rightarrow C_{n-1}(y)$ (49)
			Hyper[7+39]:	$\top \rightarrow B_2(g_2(x))$	(43)		
			Pred[...]:	$\top \rightarrow C_1(x)$	(50)		
			Hyper[9+37+50]:	$S_1(y,x) \rightarrow C_0(y)$	(51)		
			Hyper[10+38+50]:	$S_2(y,x) \rightarrow C_0(y)$	(52)		

Figure 3: Applying the Consequence-Based Calculus to $\mathcal{O}_1$

4.4.1 Ontology $\mathcal{O}_1$ form Figure 1

Let $\mathcal{O}_1$ be the ontology from Figure 1, which, for convenience, we repeat in Figure 3. The figure shows the inferences of our algorithm needed to prove $\mathcal{O}_1 \models B_0(x) \rightarrow C_0(x)$ using the cautious strategy; note that these inferences also prove $\mathcal{O}_1 \not\models B_0(x) \rightarrow \bot$. Please recall that $\mathcal{O}_1$ is an $\mathcal{EL}$ ontology. Let I be a canonical Herbrand model of $\mathcal{O}_1$.

To satisfy condition C2 of Theorem 9, the context structure is initialised with context v_0 whose core is $B_0(x)$, and the Core rule then derives (30). Next, the Hyper rule derives (31) from (4) and (30), and (32) from (5) and (30); and it derives (33) from (6) and (30), and (34) from (7) and (30). At this point, the standard hyperresolution rule (Bachmair & Ganzinger, 2001) uses (32) and (4) to derive clause $\top \rightarrow S_1(f_1(x), f_1(f_1(x)))$, which contains terms of depth two; such inferences can derive clauses with terms of arbitrary depth, which can prevent termination. Resolution-based decision procedures typically address this problem by ordering and selection restrictions mentioned in Section 3.1, but this can lead to the derivation of irrelevant clauses such as (13) and (14). Our calculus uses hyperresolution to be more goal oriented and avoid the derivation of irrelevant clauses; however, to derive only constraints about a term and its predecessor/successors and thus prevent arbitrary term nesting, variable x of a DL-clause must be matched to the variable x of context clauses.

Now consider an arbitrary term t represented in I by context v_0. Clauses (31) through (34) contain function symbols f_1 and g_1, so I may contain terms $f_1(t)$ and $g_1(t)$. These terms must be represented in the context structure, which is ensured by the Succ rule.

The Succ rule is first applied to clauses (31) and (32). At this point, the rule has no option but to introduce a fresh context v_1, but it has a choice regarding how to initialise the context's core. Clauses (31) and (32) ensure that $S_1(t, f_1(t))$ and $B_1(f_1(t))$ necessarily hold in I; since these atoms are represented in context v_1 as $S_1(y, x)$ and $B_1(x)$, the rule can initialise the core of v_1 to any subset of the conjunction $S_1(y, x) \wedge B_1(x)$. The cautious strategy initialises the core of v_1 to $B_1(x)$. Thus, $B_1(t')$ necessarily holds in I for each term t' represented by x in context v_1; however, $S_1(t'', t')$ for t'' the predecessor of t' may or may not hold in I, and so the Succ rule adds clause (37). Please note that this tautological clause is *not* redundant in our calculus: it says that $S_1(t'', t')$ is possible in I, and the clause will participate in further inferences that derive our query clause.

The Succ rule is next applied to clauses (33) and (34). At this point, however, there is a choice: the rule could introduce a fresh context, or it could reuse an existing context whose core is constructed using atoms $S_2(y, x)$ and $B_1(x)$. The core of v_1 ensures that $B_1(f_1(t))$ and $B_1(g_1(t))$ necessarily hold in I; thus, the cautions strategy can reuse v_1, so the Succ rule derives clause (38). At this point, context v_1 represents both $f_1(t)$ and $g_1(t)$ in I, which is possible only because core_{v_1} contains neither $S_1(y, x)$ nor $S_2(y, x)$. Finally, clauses (37) and (38) say that $S_1(t, f_1(t))$ and $S_2(t, g_1(t))$ may hold in I, but they (correctly) do not require $S_1(t, g_1(t))$ and $S_2(t, f_1(t))$ to hold in I. If the core of v_1 were to contain $S_1(y, x)$ and $S_2(y, x)$, then $S_1(t, g_1(t))$ and $S_2(t, f_1(t))$ would both hold in I, which would be unsound.

Next, the Core rule derives clause (39), which ensures that $B_1(t)$ holds in I for each term t represented by x in context v_1. Contexts $v_2, \ldots, v_n$ are then constructed analogously. Next, clause (47) is derived by hyperresolving (8) and (46); clause (48) is derived by hyperresolving (9), (44), and (47); and clause (49) is derived analogously. Clause (48) imposes a constraint on the predecessor context, which is propagated 'backwards' using a chain of Pred rule inferences to derive (50). Moreover, clause (49) participates in analogous inferences, which also result in deriving (50). Clause (51) is derived by hyperresolving (9), (37), and (50); and clause (52) is derived analogously. Finally, clauses (51) and (52) are propagated 'backwards' using the Pred rule to derive (53). Since the clauses of the context v_0 are 'relative' to the core of v_0, clause (53) represents $\mathcal{O}_1 \models B_0(x) \rightarrow C_0(x)$, as required. Finally, clause (50) in context v_1 implies $\mathcal{O}_1 \models B_1(x) \rightarrow C_1(x)$ and analogously for all $\mathcal{O}_1 \models B_i(x) \rightarrow C_i(x)$ for $0 \leq i \leq n$, all of which are derived in a single run of our algorithm. We next point out three important aspects of our algorithm.

First, our context structure is derived in a goal-oriented way that is reminiscent of model construction by hypertableau. However, unlike the repetitive exponentially-sized model that can be constructed using hypertableau as discussed in Section 3.1, our context structure is of polynomial size. This is possible because each context v_i and the corresponding context clauses represents all model elements at depth i.

Second, assigning the derived clauses to contexts considerably restricts the inferences. Note that each clause $\top \rightarrow C_i(x)$ is derived only in context v_i. But then, since clauses from different contexts cannot participate in an inference with DL-clause (11), our algorithm avoids the derivation of irrelevant clauses such as (13) and (14). Please note that this behaviour is similar to hypertableau, but without constructing an exponential structure. This property, we believe, is key to good performance of consequence-based calculi in practice.

Third, the names of the variables in DL-clauses determine which inferences are performed. For example, if $\mathcal{O}_1$ were extended with DL-clause $S_1(x, z_1) \rightarrow D(x)$, then $D(x)$

would be derived from (31) in context v_0 using the Hyper rule; this is analogous to reading the DL-clause as $\exists S_1.\top \sqsubseteq D$. In contrast, if $\mathcal{O}_1$ were extended by a logically equivalent DL-clause $S_1(z_1, x) \rightarrow D(z_1)$, then $D(y)$ would be derived from (37) in context v_1 using the Hyper rule, and it would be propagated as $D(x)$ to context v_0 using the Pred rule; this is analogous to reading the DL-clause as $\top \sqsubseteq \forall S_1^-.D$. On $\mathcal{EL}$ ontologies, this ensures that our calculus mimics the inferences of the calculus by Baader et al. (2005): our context structure closely corresponds to the structure constructed by that calculus, axioms $B_1 \sqsubseteq B_2$ and $B_1 \sqcap B_2 \sqsubseteq B_3$ are handled analogously to the Hyper rule, axioms $B_1 \sqsubseteq \exists S.B_2$ are handled by an inference rule that corresponds to an application of Hyper rule followed by the Succ rule, and axioms $\exists S.B_1 \sqsubseteq B_2$ are handled by an inference rule that corresponds to an application of Hyper rule followed by the Pred rule.

4.4.2 Ontology $\mathcal{O}_2$ form Section 3.3

Let $\mathcal{O}_2$ be the ontology from Figure 2, which, for convenience, we repeat in Figure 4. The figure shows how our calculus proves $\mathcal{O}_2 \models B_0(x) \rightarrow B_4(x)$ using the eager strategy. We use a context term order according to which the maximal literal of each derived non-Horn clause is underlined as shown in the figure. Let I be a Herbrand model of $\mathcal{O}_2$.

Our calculus again introduces context v_0 and initialises its core to $B_0(x)$. The Core rule next initialises v_0 with (54), ensuring that I contains a ground term for which B_0 holds. Next, the calculus derives (55) and (56) using the Hyper rules.

The Succ rule next ensures that, due to the function symbol f_1 in clauses (55) and (56), the context structure contains an appropriate successor of context v_0. To see which information is relevant to the successor, note that DL-clause (25) contains atoms $B_1(x)$ and $S(x, z_i)$ in its body, and that z_i can be mapped to a predecessor or a successor of x; thus, a context in which hyperresolution is applied to (25) will be interested in information about its predecessors. This is reflected by adding $B_1(x)$ and $S(x, y)$ to the set $\mathsf{Su}(\mathcal{O})$, which determines the information to be propagated to the successor. By the eager strategy, the Succ rule introduces context v_1 and sets its core to $S(x, y) \wedge B_1(x)$.

The Hyper rule next introduces clauses (60) through (63), at which point there is sufficient information to apply hyperresolution to (25) to derive (64). Please note that clause (58) is required in this inference step.

Equality $f_3(x) \approx f_2(x)$ in clause (64) is handled using paramodulation—that is, occurrences of $f_3(x)$ are replaced by $f_2(x)$. Thus, paramodulating clause (64) into clause (63) produces clause (65). Moreover, replacing the term $f_3(x)$ with the term $f_2(x)$ in clause (62) produces a clause that is redundant due to clause (60), so the inference is not applicable.

Clauses (60), (61), and (65) contain the function symbol f_2, so the Succ rule introduces context v_2. Observe that atom $B_2(f_2(x))$ occurs in clause (61) as the sole atom, and thus atom $B_2(t)$ holds in I for each ground term t represented by v_2. Hence, $B_2(x)$ can be added to the core of v_2. In contrast, atom $B_3(f_2(x))$ occurs in clause (65) with more disjuncts; thus, $B_3(f_2(t))$ may hold in I, but that is not necessary since other disjuncts might satisfy clause (65). Atom $B_3(x)$ thus cannot be added to the core of v_2, so the Succ rule introduces clause (67), which contains $B_3(x)$ in the body. This clause is again *not* redundant: it says that $B_3(f_2(t))$ may be present in I, which allows the Hyper rule to derive (70) and obtain further constraints that must be satisfied for $f_2(t)$ in case $B_3(f_2(t))$ is indeed present.

Ontology $\mathcal{O}_2$

$B_0 \sqsubseteq \exists S^-.B_1$	$\rightsquigarrow$	$B_0(x) \to S(f_1(x), x)$	(16)
		$B_0(x) \to B_1(f_1(x))$	(17)
$B_1 \sqsubseteq \exists S.B_2$	$\rightsquigarrow$	$B_1(x) \to S(x, f_2(x))$	(18)
		$B_1(x) \to B_2(f_2(x))$	(19)
$B_1 \sqsubseteq \exists S.B_3$	$\rightsquigarrow$	$B_1(x) \to S(x, f_3(x))$	(20)
		$B_1(x) \to B_3(f_3(x))$	(21)
$B_2 \sqsubseteq B_4$	$\rightsquigarrow$	$B_2(x) \to B_4(x)$	(22)
$B_3 \sqsubseteq B_4$	$\rightsquigarrow$	$B_3(x) \to B_4(x)$	(23)
$B_2 \sqcap B_3 \sqsubseteq \bot$	$\rightsquigarrow$	$B_2(x) \wedge B_3(x) \to \bot$	(24)
$B_1 \sqsubseteq \leq 2.S$	$\rightsquigarrow$	$B_1(x) \wedge S(x, z_1) \wedge S(x, z_2) \wedge S(x, z_3) \to z_1 \approx z_2 \vee z_1 \approx z_3 \vee z_2 \approx z_3$	(25)

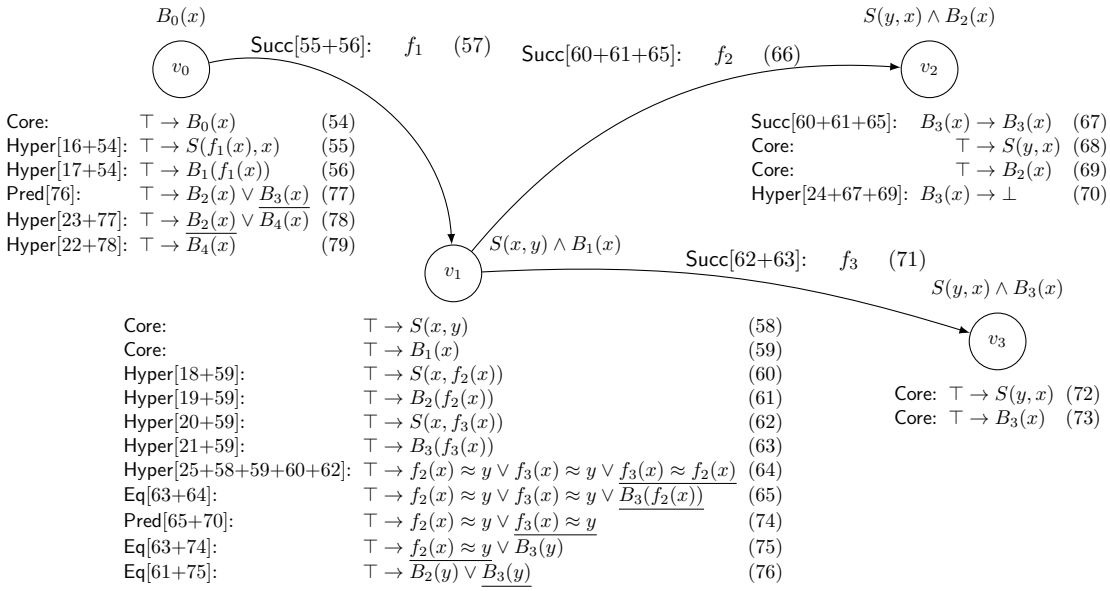


Figure 4: Applying the Consequence-Based Calculus to $\mathcal{O}_2$

Clause (70) essentially says '$B_3(f_2(x))$ should not hold in the predecessor', which the Pred rule propagates to v_1 as clause (74); this can be understood as hyperresolution of (65) and (70) while observing that $f_2(x)$ in context v_1 is represented as x in context v_2.

Two further paramodulation steps derive clause (76), which essentially says 'the predecessor must satisfy $B_2(x)$ or $B_3(x)$'. Now DL-clauses (22) and (23) contain $B_2(x)$ and $B_3(x)$, respectively, in their bodies, which are represented in v_1 as $B_2(y)$ and $B_3(y)$. To identify these atoms as relevant for predecessors, set $\mathsf{Pr}(\mathcal{O})$ contains $B_2(y)$ and $B_3(y)$, which in turn allows the Pred rule to derive clause (77).

Two further steps derive clause (79) in context v_0. This would not be possible if $B_4(x)$ were maximal in (78); thus, condition C2 of Theorem 9 requires all atoms in the head of a

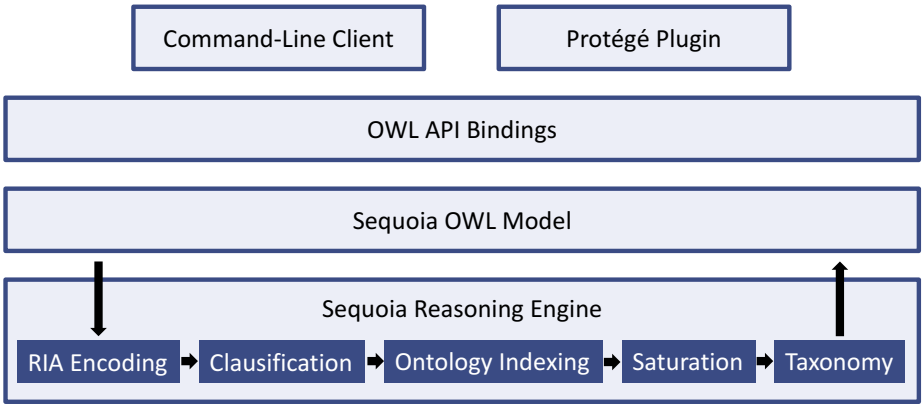


Figure 5: The components of the Sequoia reasoner and data flow during classification

query clause to be smallest in the context term order. Similarly, clause (76) is relevant for context v_0, but it could not be derived if atom $B_3(y)$ were maximal in (75); thus, condition 5 of Definition 3 requires all atoms from $\mathsf{Pr}(\mathcal{O})$ to be smallest in the context term order.

5. Implementation of Sequoia

We implemented our calculus in a new reasoner called *Sequoia*.[3] The system handles the DL $\mathcal{SRIQ}$ and can thus process all constructs of OWL 2 DL apart from nominals and datatypes; moreover, ABoxes are not supported since the system can currently only decide concept satisfiability and concept subsumption, as well as classify an ontology. The system is written in Scala. In this section, we describe the system's architecture and discuss certain issues that we identified as critical to the system's performance.

5.1 System Architecture

Figure 5 shows the architecture of Sequoia. The system can be used on the command line, as a Protégé plug-in (Knublauch, Fergerson, Noy, & Musen, 2004), or as a library implementing the OWL API (Horridge & Bechhofer, 2011). The *Reasoning Engine* is the core component of Sequoia, and it comprises several steps that communicate as shown by the arrows in the figure. It is given as input an OWL API ontology. The *RIA Encoding* step uses the algorithm by Simančík (2012) to transform role inclusion axioms in the input into $\mathcal{ALCHIQ}^+$ axioms. The *Clausification* step transforms these axioms into a set of DL-clauses, which are next indexed in the *Ontology Indexing* step. The *Saturation* step implements the core reasoning algorithm from Section 4, and in the rest of this section we discuss several implementation techniques. Finally, the *Taxonomy* step collects the subsumption information and produces the (transitively reduced) concept hierarchy.

3. `http://github.com/andrewdbate/Sequoia/`

5.2 Saturation Implementation

To classify an ontology, Sequoia proceeds as outlined in Section 4.3: it creates a context structure $\mathcal{D}$ that contains, for each atomic concept B, a context with core $B(x)$ where all atomic concepts are incomparable; it saturates $\mathcal{D}$ using the inference rules of the calculus; and it reads off the subsumptions from the saturated context structure. In this section we discuss the saturation procedure of Sequoia, which is inspired by the algorithm used by first-order resolution-based theorem provers Prover9,[4] Otter (McCune & Wos, 1997), and an early version of Vampire (Riazanov & Voronkov, 2003).

The pseudocode of our saturation algorithm is shown in Algorithms 2 and 3. The procedure is given the input ontology $\mathcal{O}$ and the context structure $\mathcal{D}$ to be saturated. To simplify the presentation, we assume that the Succ rule is instantiated for the selected expansion strategy. For reasons we describe shortly, the Ineq and the Elim rule are applied as special cases. Moreover, the Hyper, Eq, and Factor rule are said to be *local* since they involve context clauses associated with just one context.

For each context v, the procedure uses three sets of *unprocessed* clauses: L_v accumulates the clauses derived in context v to which the local rules must be applied, and P_v and U_v do the same for the Pred and the Succ rule, respectively. Moreover, for each context v, a Boolean flag I_v records whether the context has been initialised. Finally, for each context v, set E_v stores tuples of the form $\langle u, f \rangle$ for u a context and f a function symbol, indicating that the Pred rule may need to be applied to an edge $\langle u, v, f \rangle$. For each context v, these sets are initialised in line 2, the calls to Derive(v, D) in line 3 'copy' all clauses from $\mathcal{S}_v$ into L_v, P_v, and U_v, and finally set $\mathcal{S}_v$ is emptied in 4; these steps ensure that all inferences between the clauses in $\mathcal{S}_v$ are taken into account.

The algorithm next enters a loop (lines 5 to 34) in which it iteratively applies inferences. The rules are applied in four stages: (i) Core, (ii) Hyper, Eq, and Factor, (iii) Pred, and (iv) Succ. All rules in a stage are exhaustively applied to a context before moving on to the next stage. Our experience has shown that delaying the Pred and Succ rules as long as possible is often beneficial. Moreover, the Eq and the Factor are not applicable if $\mathcal{O}$ is an $\mathcal{EL}$ ontology, so this rule application strategy satisfies the conditions of Proposition 12.

Lines 6 to 9 initialise a context. In particular, line 7 applies the Core rule, and line 8 applies the Hyper rule for the DL-clauses of $\mathcal{O}$ whose body is empty.

Lines 10 to 14 apply the local rules according to the earlier mentioned precedence. To saturate a context v, a clause C is selected and removed from L_v in line 11; in Section 5.2.2 we describe the heuristics that Sequoia uses in this step. Clause C is next added to $\mathcal{S}_v$ (line 12). Next, all local inference rules are applied to C and the clauses in $\mathcal{S}_v$ (lines 13 to 14). Since inferences can involve more than one clause and the set $\mathcal{S}_v$ can be large, the participating clauses are identified using indexes that we describe in Section 5.3. Finally, each conclusion D is passed to the auxiliary procedure Derive which tries to simplify D and eliminate redundant clauses, and eventually add the conclusion to L_v, P_v, and U_v and thus schedule it for future processing; we describe this procedure in Section 5.2.1.

Lines 15 to 20 apply the Pred rule in a similar way. Unlike in the previous paragraph, C need not be added to $\mathcal{S}_v$ because C is added to L_v in line 5 and set L_v is processed fully before processing P_v. Moreover, lines 17 to 18 compute the consequences using edges

4. http://www.cs.unm.edu/~mccune/prover9/

Algorithm 2 The saturation algorithm of Sequoia

Input: $\mathcal{O}$: the input ontology
$\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathsf{core}, \succcurlyeq, \succ \rangle$: the context structure to saturate

for each $v \in \mathcal{V}$ **do**
  $\mathsf{L}_v := \mathsf{P}_v := \mathsf{U}_v := \mathsf{E}_v := \emptyset, \quad \mathsf{I}_v := \mathit{false}$
  for each $D \in \mathcal{S}_v$ **do** Derive(v, D)
  $\mathcal{S}_v := \emptyset$
loop
  if there exists $v \in \mathcal{V}$ with $\mathsf{I}_v = \mathit{false}$ **then**
    for each $A \in \mathsf{core}_v$ **do** Derive$(v, \top \to A)$
    for each $\top \to \Delta \in \mathcal{O}$ **do** Derive$(v, \top \to \Delta)$
    $\mathsf{I}_v := \mathit{true}$
  else if there exists $v \in \mathcal{V}$ with $\mathsf{L}_v \neq \emptyset$ **then**
    select and remove a clause C from L_v
    add C to $\mathcal{S}_v$
    for each $r \in \{\mathsf{Hyper}, \mathsf{Eq}, \mathsf{Factor}\}$ in that order **do**
      for each $D \in$ ApplyLocalRule$(\mathcal{O}, \mathcal{D}, r, v, C)$ **do** Derive(v, D)
  else if there exists $v \in \mathcal{V}$ with $\mathsf{P}_v \neq \emptyset$ **then**
    select and remove a clause C from P_v
    for each $\langle u, v, f \rangle \in \mathcal{E}$ **do**
      for each $D \in$ ApplyPredTo$(\mathcal{D}, u, v, f, C)$ **do** Derive(u, D)
    for each $\langle v, u, f \rangle \in \mathcal{E}$ **do**
      for each $D \in$ ApplyPredFrom$(\mathcal{D}, v, u, f, C)$ **do** Derive(v, D)
  else if there exists $v \in \mathcal{V}$ with $\mathsf{E}_v \neq \emptyset$ **then**
    select and remove a tuple $\langle u, f \rangle$ from E_v
    for each $C \in \mathcal{S}_v$ **do**
      for each $D \in$ ApplyPredTo$(\mathcal{D}, u, v, f, C)$ **do** Derive(u, D)
  else if there exists $u \in \mathcal{V}$ with $\mathsf{U}_u \neq \emptyset$ **then**
    select and remove a clause C from U_u
    for each $\langle v, f, \mathsf{core}', \succ', \mathsf{Clauses} \rangle \in$ ApplySucc$(\mathcal{D}, u, C)$ **do**
      if $v \notin \mathcal{V}$ **then**
        add v to $\mathcal{V}$
        $\mathsf{core}_v := \mathsf{core}', \quad \succ_v := \succ', \quad \mathsf{L}_v := \mathsf{P}_v := \mathsf{U}_v := \emptyset, \quad \mathsf{I}_v := \mathit{false}$
      if $\langle u, v, f \rangle \notin \mathcal{E}$ **then add** $\langle u, v, f \rangle$ to $\mathcal{E}$, and also add $\langle u, f \rangle$ to E_v
      for each $D \in \mathsf{Clauses}$ **do** Derive(v, D)
  else
    return $\mathcal{D}$

Algorithm 3 Derive(v, D)

1: $D :=$ SimplifyClause$(D, \mathcal{S}_v \cup \mathsf{L}_v)$
2: **if** $D \neq$ *null* and no $D' \in \mathcal{S}_v \cup \mathsf{L}_v$ exists such that Redundant(D, D') **then**
3: **remove** from $\mathcal{S}_v$, P_v, and U_v each $D' \in \mathcal{S}_v$ such that Redundant(D', D)
4: **remove** from L_v, P_v, and U_v each $D' \in \mathsf{L}_v$ such that Redundant(D', D)
5: **add** D to L_v, P_v, and U_v

pointing towards v for the case when C plays the role of the clause $\bigwedge_{i=1}^{m} A_i \to \bigvee_{i=m+1}^{m+n} L_i$ from Table 2. Finally, lines 19 to 20 compute the consequences using edges pointing from v for the case when C plays the role of a clause $\Gamma_i \to \Delta_i \vee A_i\sigma$ from Table 2.

Lines 25 to 32 apply the Succ rule to a clause C. The main difference to the previous cases is that the rule application may involve extending the context structure. Thus, ApplySucc$(\mathcal{D}, u, C)$ returns a set of tuples of the form $\langle v, f, \mathsf{core}', \succ', \mathsf{Clauses}\rangle$ where v is the (possibly fresh) context used to satisfy the rule via edge $\langle u, v, f\rangle$; core' and $\succ'$ are the core and the term order of v as returned by the expansion strategy; and Clauses contains clauses that must be derived in context v. The procedure extends the sets of contexts (lines 28 to 30) and edges (line 31) as needed, and then derives the relevant clauses (line 32). Note that if the context term order of u ensures that each clause has at most one maximal atom with a function symbol, then at most one $\langle v, f, \mathsf{core}', \succ', \mathsf{Clauses}\rangle$ is returned in line 27.

To understand the rationale behind lines 21 to 24, assume that a context structure contains disconnected contexts u and v where P_u and P_v are both empty (i.e., the Pred rule has been applied exhaustively to both contexts). Now if the Succ rule introduces an edge from u to v, the Pred rule could become applicable to clauses in $\mathcal{S}_u$ and $\mathcal{S}_v$; however, since $\mathsf{P}_u = \mathsf{P}_v = \emptyset$, this inference will not take place in lines 15 to 20. To ensure that such inferences are not missed, whenever an edge $\langle u, v, f\rangle$ is added to $\mathcal{E}$ in line 31, $\langle u, f\rangle$ is also added to E_v. Subsequently, such $\langle u, f\rangle$ is retrieved from E_v in line 22, and the Pred rule is applied to the edge $\langle u, v, f\rangle$ and each clause $C \in \mathcal{S}_v$.

Algorithm 2 can be easily parallelised. Lines 6 to 14 involve just one context v, so they can be applied on separate threads independently from other contexts. Moreover, when a thread for context v derives in lines 15 to 32 a clause D that should be added to L_u, P_u, or U_u for a different context u, the thread for context v can just send a message to the thread for context u that D has been derived in context u, and then the latter thread can update L_u, P_u, and U_u without any locking.

5.2.1 Simplification and Redundancy Elimination

Before inserting a conclusion $D = \Gamma \to \Delta \in$ into the target set $\mathcal{S}_v$ of context clauses, Algorithm 3 tries to simplify D and eliminate redundant clauses. Since each derived clause is contained in $\mathcal{S}_v$ or L_v, both of these steps are performed w.r.t. the clauses in $\mathcal{S}_v \cup \mathsf{L}_v$.

In the simplification step, line 1 calls SimplifyClause$(D, \mathcal{S}_v \cup \mathsf{L}_v)$, which tries to transform D into a simpler clause using the techniques described below.

- The Ineq rule is applied to D as a simplification rule—that is, each literal of the form $s \not\approx s$ is deleted from Δ. The result of the Ineq rule always makes the original clause redundant, which is why the rule is applied in the simplification step.

- If $\{s \approx s',\ s \not\approx s'\} \subseteq \Delta$ or $s \approx s \in \Delta$ holds for some terms s and s', clause D is discarded (by returning *null*) in accordance with condition 1 of Definition 4.
- A simplified variant of *equational literal cutting* (Schulz, 2013, 2002) is applied: if $s \approx t \in \Delta$ and $\top \rightarrow s \not\approx t \in \mathcal{S}_v \cup \mathsf{L}_v$, then $s \approx t$ is deleted from Δ; also, if $s \not\approx t \in \Delta$ and $\top \rightarrow s \approx t \in \mathcal{S}_v \cup \mathsf{L}_v$, then $s \not\approx t$ is deleted from Δ. These optimisations are known in the literature as *negative* and *positive simplify-reflect* (Schulz, 2002), respectively, and they correspond to a combination of the Eq and Elim rules.

Redundancy elimination uses an auxiliary procedure $\textsc{Redundant}(\Gamma_1 \rightarrow \Delta_1, \Gamma_2 \rightarrow \Delta_2)$, which returns *true* if $\Gamma_2 \subseteq \Gamma_1$ and $\Delta_2 \subseteq \Delta_1$; in such a case, clause $\Gamma_2 \rightarrow \Delta_2$ is said to be *stronger* than $\Gamma_1 \rightarrow \Delta_1$, and conversely $\Gamma_1 \rightarrow \Delta_1$ is *weaker* than $\Gamma_2 \rightarrow \Delta_2$.

Line 2 implements *forward redundancy elimination*, which involves checking whether set $\mathcal{S}_v \cup \mathsf{L}_v$ contains a clause D' that is stronger than D. Finally, if the simplified clause is not forward redundant, *backward redundancy elimination* (lines 3 and 4) deletes from $\mathcal{S}_v$, L_v, P_v, and U_v all clauses that are weaker than D. Both forward and backward redundancy elimination require searching potentially large clause sets so, to optimise this search, Sequoia maintains redundancy indexes that we describe in Section 5.3.5.

5.2.2 Selecting Clauses

To reduce the number of clauses retained during forward redundancy elimination and increase the number of clauses removed during backward redundancy elimination, Horn clauses with empty bodies should be preferred to Horn clauses with nonempty bodies, Horn clauses should be preferred to non-Horn clauses, and shorter clauses should be preferred to longer clauses. To implement this policy, each clause C is an assigned an integer priority according to these rules, and sets L_v, P_v, and U_v are implemented as custom array-based data structures that are optimised for addition and removal at both ends.

5.2.3 Discussion

Practical experiments have shown that it is beneficial to spend time simplifying and removing redundant clauses. This can considerably reduce the memory consumption (thus possibly avoiding memory exhaustion) and can lead to important simplifications being applied earlier (thus reducing the total number of generated clauses). Furthermore, our experience shows that, for ontologies encountered in practice, the contexts generated by our calculus are usually satisfiable. Hence, eager simplification and redundancy elimination are both important because our system usually fully saturates set of clauses in each context.

We could have used a saturation procedure where a clause is simplified or used to simplify clauses only when it is selected in line 11, 16, or 26. This form of delayed simplification is known as the Discount loop (Riazanov & Voronkov, 2003) and it is used in the KAON2 reasoner (Hustadt et al., 2007). In contrast, the Otter loop (McCune & Wos, 1997; Weidenbach et al., 1996) potentially introduces important simplifications earlier and thus spends much more time in simplifying clauses. Neither variant is uniformly better, so we tested both and selected the Otter loop since it exhibited the best overall performance.

5.3 Clause Indexing Data Structures

Like first-order theorem provers, Sequoia uses several indexes to efficiently identify clauses needed to apply an inference or a simplification rule. Ramakrishnan, Sekar, and Voronkov (2001) and Graf (1996) present a comprehensive survey of the indexing techniques used in first-order theorem provers. While these techniques have proved themselves in practice, the restricted structure of our clauses allow us to use simpler, yet still efficient techniques.

5.3.1 Encoding Names, Terms and Literals using Integers

During clausification (see Figure 5), each atomic concept, atomic role, and term is assigned a unique integer identifier (UID). UIDs are assigned sequentially, so we can use perfect hashing in indexes whose keys are names or terms. Moreover, if y and x are assigned the smallest UIDs, a context term order on a-terms can be obtained by comparing the UIDs.

We represent each literal using a 64-bit integer that contains the type of the literal (a unary atom, a binary atom, an equality, or an inequality), a flag specifying whether the literal is occurring in the body or the head of a clause, the identifier of the predicate for unary and binary atoms, and the terms occurring in the literal. Terms of equalities and inequalities are sorted according to the term order—that is, $s \succ t$ holds in $s \approx t$ and $s \not\approx t$.

5.3.2 Unification Indexing for the Hyper Rule

To speed up the application of the Hyper rule, Sequoia maintains several indexes that can quickly identify the clauses that can participate in the rule. To facilitate indexing, we assign to each atom A a *unifier pattern* defined as follows:

$$\mathsf{pattern}(A) = \begin{cases} B(x) & \text{if } A = B(x) \\ B(*) & \text{if } A = B(t) \text{ and } t \neq x \\ S(x,x) & \text{if } A = S(x,x) \\ S(x,*) & \text{if } A = S(x,t) \text{ and } t \neq x \\ S(*,x) & \text{if } A = S(t,x) \text{ and } t \neq x. \end{cases}$$

This definition enjoys the following property: for all atoms A_1 and A_2 for which a substitution σ exists such that $A_1 = A_2\sigma$ and $\sigma(x) = x$ (as required by the preconditions of the Hyper rule), we have $\mathsf{pattern}(A_1) = \mathsf{pattern}(A_2)$.

During the ontology indexing phase, ontology $\mathcal{O}$ is indexed using two indexes. Index $\mathsf{ontologyIndex}_1$ is a hash table that maps a unifier pattern p to the set of all DL-clauses of $\mathcal{O}$ that contain in their body an atom A whose unifier pattern is p:

$$\mathsf{ontologyIndex}_1[p] = \{\, \Gamma \to \Delta \in \mathcal{O} \mid \exists A \in \Gamma \text{ such that } p = \mathsf{pattern}(A) \,\}$$

Moreover, $\mathsf{ontologyIndex}_2$ is the set of all DL-clauses of $\mathcal{O}$ whose body is empty:

$$\mathsf{ontologyIndex}_2 = \{\, C \mid C = \Gamma \to \Delta \in \mathcal{O} \text{ with } \Gamma = \top \,\}$$

Finally, for each context $v \in \mathcal{V}$, we index $\mathcal{S}_v$ using a hash table $\mathsf{hyperIndex}_v$ that maps a unifier pattern p to the set of all clauses of $\mathcal{S}_v$ that contain in the head a maximal atom whose unifier pattern is p:

$$\mathsf{hyperIndex}_v[p] = \{\, \Gamma \to \Delta \in \mathcal{S}_v \mid \exists A \in \Delta \text{ such that } p = \mathsf{pattern}(A) \text{ and } \Delta \setminus \{A\} \not\succeq_v A \,\}$$

Indexes $\mathsf{ontologyIndex}_1$ and $\mathsf{ontologyIndex}_2$ contain the DL-clauses in $\mathcal{O}$ and are thus immutable, whereas $\mathsf{hyperIndex}_v$ must be updated whenever $\mathcal{S}_v$ changes.

Index $\mathsf{ontologyIndex}_2$ is used in line 8 of Algorithm 2. Moreover, this index is also used to apply the Hyper rule to a head atom A_1 of a clause C in line 13 as follows. First, we query index $\mathsf{ontologyIndex}_1[\mathsf{pattern}(A)]$ to identify each DL-clause $\Gamma \to \Delta$ and body atom $A'_1 \in \Gamma$ for which there exists a substitution σ_1 such that $A_1\sigma_1 = A'_1$ and $\sigma_1(x) = x$. For each such match, we consider each atom $A'_i \in \Gamma \setminus \{A'_1\}$ and query $\mathsf{hyperIndex}_v[\mathsf{pattern}(A'_i)]$ to identify all premises containing a head atom A_i for which there exists a substitution σ_i such that $A_i\sigma_i = A'_i$ and $\sigma_i(x) = x$. For each thus obtained set of matching premises where σ_i are not contradictory (i.e., variables are mapped by all σ_i in the same way), we apply the Hyper rule with the substitution $\sigma = \bigcup_i \sigma_i$ and derive the corresponding clause.

Such an approach can be inefficient if a predicate occurs in many DL-clause bodies since then we iterate over a large set $\mathsf{ontologyIndex}_1[\mathsf{pattern}(A)]$ for each distinct A. Reasoners often optimise application of the Hyper rule (Baader, Lutz, & Suntisrivaraporn, 2006; Sertkaya, 2011; Kazakov et al., 2014) by ensuring that DL-clauses contain at most two body atoms and then constructing an additional index whose key is a pair of access patterns corresponding to the two body atoms; then, if $\mathcal{S}_v$ is smaller than $\mathsf{ontologyIndex}_1[\mathsf{pattern}(A)]$, the Hyper rule is applied by iterating over $\mathcal{S}_v$ and identifying the relevant DL-clauses in this additional index. The main obstacle to applying this approach in our setting is that, due to number restrictions, DL-clauses cannot be restricted to just two body atoms only; thus, we leave a further investigation of this technique for our future work.

5.3.3 Context Clause Indexing for the Pred Rule

To speed up the application of the Pred rule, for each context $v \in \mathcal{V}$, we index $\mathcal{S}_v$ using hash tables $\mathsf{predBodyIndex}_v$ and $\mathsf{predHeadIndex}_v$ that map an atom A to the sets of all clauses of $\mathcal{S}_v$ that contain A in the body and head, respectively:

$$\mathsf{predBodyIndex}_v[A] = \{\, \Gamma \to \Delta \in \mathcal{S}_v \mid A \in \Gamma \text{ and } \Delta \subseteq \mathsf{Pr}(\mathcal{O}) \,\}$$
$$\mathsf{predHeadIndex}_v[A] = \{\, \Gamma \to \Delta \in \mathcal{S}_v \mid A \in \Delta \text{ and } \Delta \setminus \{A\} \not\succeq_v A \,\}$$

These indexes are maintained whenever a clause is added to or removed from $\mathcal{S}_v$. To apply the Pred rule to a head atom A of a clause C, we query $\mathsf{predBodyIndex}_v[A']$ for atom A' obtained from A in a way that enables the rule; then, for each clause $\Gamma \to \Delta$ from this set and each atom $A'' \in \Gamma$, we query $\mathsf{predHeadIndex}_v[A''']$ where A''' is obtained from A''.

5.3.4 Context Clause Indexing for the Eq Rule

To speed up the application of the Eq rule, for each context $v \in \mathcal{V}$, we index $\mathcal{S}_v$ using the hash table $\mathsf{eqIndex}_v$ that maps a term s to the following set of all clauses of $\mathcal{S}_v$:

$$\begin{aligned}\mathsf{eqIndex}_v[s] = \{\, \Gamma \to \Delta \in \mathcal{S}_v \mid \exists L \in \Delta \text{ such that } \Delta \setminus \{L\} \not\succeq_v L \text{ and} \\ \text{(i) } L \text{ is an atom, } s \text{ is of the form } f(x), \text{ and } s \text{ occurs in } L, \text{ or} \\ \text{(ii) } L \text{ is of the form } s \approx t \text{ or } s \not\approx t, \text{ and } s \succ_v t \,\}\end{aligned}$$

Note that a term s in a head atom can participate in the Eq rule only if it is of the form $f(x)$, so we only index such terms. This index must be maintained whenever $\mathcal{S}_v$ changes.

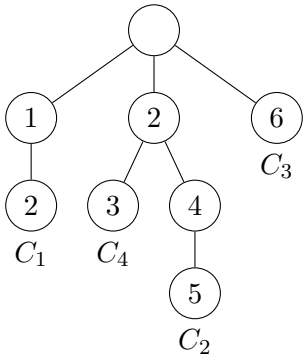


Figure 6: Example redundancy index

5.3.5 Redundancy Indexes

Redundancy indexes are used in two related problems: given a clause C and a set of clauses N, *forward redundancy elimination* determines whether N contains a clause that is stronger than C, and *backward redundancy elimination* retrieves all clauses of N that are weaker than C. Redundancy indexing is difficult since the body and the head of a clause must be matched to a subset of the body and the head of another clause. There are exponentially many such subsets, which makes designing an efficient indexing structure challenging. Our solution is inspired by *feature vector indexing* from the theorem prover E (Schulz, 2004). The restricted structure of context clauses allows for *perfect indexing*, which returns exactly the required set of clauses; in contrast, feature vector indexes in first-order provers return a set of candidate clauses that must additionally be subjected to a redundancy test.

Sequoia uses a single index per context v for both types of redundancy elimination. The index contains all clauses in $\mathcal{S}_v \cup \mathsf{L}_v$. To index a clause $C = \bigwedge_{i=1}^{m} L_i \rightarrow \bigvee_{i=m+1}^{n} L_i$, we construct a sequence of integers $k_1, \ldots, k_n$ where k_i is the integer representation of L_i (see Section 5.3.1). We then sort this sequence into ascending order and use it as the key to insert the clause C into a trie data structure. Since we will always eliminate redundant clauses, the trie never contains distinct clauses $\Gamma_1 \rightarrow \Delta_1$ and $\Gamma_2 \rightarrow \Delta_2$ such that $\Gamma_1 \subseteq \Gamma_2$ and $\Delta_1 \subseteq \Delta_2$; consequently, clauses are always stored in the leaf nodes of the trie.

For example, consider clauses (80) to (83) and, for simplicity, assume that each literal $A_i(x)$ is represented as the integer i. Figure 6 shows the corresponding redundancy index.

$$C_1 = \top \rightarrow A_1(x) \vee A_2(x) \tag{80}$$
$$C_2 = \top \rightarrow A_2(x) \vee A_4(x) \vee A_5(x) \tag{81}$$
$$C_3 = \top \rightarrow A_6(x) \tag{82}$$
$$C_4 = \top \rightarrow A_2(x) \vee A_3(x) \tag{83}$$

We next present algorithms that use the redundancy index to support forward and backward redundancy elimination. The algorithms use the following operations that take a sequence of integers $key = k_1, \ldots, k_n$.

- IsEmpty(key) returns *true* if key is the empty sequence (i.e., if $n = 0$).

Algorithm 4 Forward Redundancy Elimination

```
1: procedure ContainsStronger(node, key)
2:     if IsLeaf(node) then
3:         return true
4:     else if IsEmpty(key) then
5:         return false
6:     else
7:         for each subkey ∈ Suffixes(key) do
8:             child := Child(node, Head(subkey))
9:             if child ≠ null and ContainsStronger(child, Tail(subkey)) then
10:                return true
11:        return false
```

- Suffixes(key) returns the set containing each sequence $k_i, \ldots, k_n$ for $1 \leq i \leq n$.
- Head(key) returns the integer k_1.
- Tail(key) returns the sequence $k_2, \ldots, k_n$.

Moreover, the algorithms also use the following operations that take a trie node.

- IsLeaf($node$) returns *true* if *node* is a leaf node.
- Child($node, k$) returns the child of *node* associated with literal integer k, or *null* if no such child exists.
- Children($node$) returns the set of all children of *node*.
- Literal($node$) returns the literal integer associated with *node*.
- Clause($node$) returns the clause associated with *node* (assuming the latter is a leaf).

Forward Redundancy Elimination To check whether the redundancy index contains a clause that is stronger than a clause C, we convert C into a sequence *key* as described earlier, and we search the trie by calling ContainsStronger($root, key$) for *root* the root of the trie as described in Algorithm 4. The crux of the algorithm is in line 7, where, given a sequence $key = k_1, \ldots, k_n$, the algorithm attempts to find, for each k_i with $1 \leq i \leq n$, a child of *node* labelled with k_i, thus skipping over gaps in the clause.

Backward Redundancy Elimination To retrieve from the redundancy index all clauses that are weaker than a clause C, we convert C into a sequence *key* as described earlier, and we search the trie by calling SelectWeaker($root, key$) for *root* the root of the trie as described in Algorithm 5. If the procedure is called with *node* and an empty sequence, in lines 3 to 9 it collects all clauses underneath *node*. Otherwise, in lines 10 to 14 the algorithm examines each child of *node*. If the child is labelled with a literal index larger than the first element of *key*, then no clause in the subtree under *node* is weaker than C. If the child is labelled with a literal index smaller than the first element of *key*, then in line 12 the algorithm tries to match *key* in the subtree under *node*. Finally, if the child is labelled with a literal index equal to the first element of *key*, then in line 14 the algorithm tries to match the rest of *key* in the subtree under *node*.

Algorithm 5 Backward Redundancy Elimination

```
1: procedure SelectWeaker(node, key)
2:     R := ∅
3:     if IsEmpty(key) then
4:         if IsLeaf(node) then
5:             R := {Clause(node)}
6:         else
7:             for each child ∈ Children(node) do
8:                 R := R ∪ SelectWeaker(child, key)
9:     else
10:        for each child ∈ Children(node) do
11:            if Literal(child) < Head(key) then
12:                R := R ∪ SelectWeaker(child, key)
13:            else if Literal(child) = Head(key) then
14:                R := R ∪ SelectWeaker(child, Tail(key))
15:    return R
```

5.4 Ordering the Context Terms

Definition 3 captures only the conditions on the term order $\succ$ that are necessary for the proof of Theorem 9. Hence, there is considerable freedom in choosing the order, which can significantly affect the performance. For example, we can extend the order $\succcurlyeq$ to the symbols of sort p and then, for atoms $B_1(t_1)$ and $B_2(t_2)$ outside $\mathsf{Pr}(\mathcal{O})$, we let $B_1(t_1) \succ B_2(t_2)$ if

1. $B_1 \succcurlyeq B_2$, or $B_1 = B_2$ and $t_1 \succ t_2$; or
2. $t_1 \succ t_2$, or $t_1 = t_2$ and $B_1 \succcurlyeq B_2$.

If we chose rule 2, then $A(f(x)) \succ B(x)$ holds regardless of the relative order of A and B, whereas this is not the case for rule 1. Rule 2 should typically be preferable to rule 1: each unary predicate occurring in the body of a DL-clause contains only variable x, and the substitution σ used in the Hyper rule must satisfy $\sigma(x) = x$; hence, rule 2 ensures that no clause containing an atom of the form $A(f(x))$ can participate in an inference with the Hyper rule, which can considerably reduce the number of inferences. This principle can be applied to all predicates, regardless of their arity. Rule 2 can be enforced by defining $\succcurlyeq$ such that all function symbols are larger than all predicate symbols.

As we discussed in Section 2.4, normalisation of an ontology can introduce fresh atomic concepts, and it is beneficial to make these concepts smallest in the order $\succcurlyeq$ of function symbols. To see why, let $\mathcal{O}_3$ be the ontology containing the following DL-clauses, where $T_1, \ldots, T_n$ are fresh predicates introduced during normalisation:

$$\rightarrow T_1(x) \vee \ldots \vee T_n(x) \tag{84}$$

$$T_i(x) \rightarrow A_i(x) \qquad \text{for } 1 \leq i \leq n \tag{85}$$

$$T_i(x) \rightarrow B_i(x) \qquad \text{for } 1 \leq i \leq n \tag{86}$$

Note that fresh predicates occur in both heads and bodies of clauses, as shown in $\mathcal{O}_3$. Assume now that $\succcurlyeq$ satisfies $A_1 \succcurlyeq B_1 \succcurlyeq A_2 \succcurlyeq B_2 \succcurlyeq \ldots \succcurlyeq A_n \succcurlyeq B_n$ and $T_1 \succcurlyeq T_2 \succcurlyeq \ldots \succcurlyeq T_n$,

and furthermore assume that we wish to check the consistency of $\mathcal{O}_3$; for simplicity, we assume we use the trivial strategy so all clauses occur in a single context. If we assume that $\succcurlyeq$ satisfies $T_n \succcurlyeq A_1$ (i.e., if the fresh predicates introduced during normalisation are largest in the function symbol order), then our calculus derives 2^n clauses. In contrast, if $\succcurlyeq$ satisfies $B_n \succcurlyeq T_1$ (i.e., the fresh predicates are smallest in the function symbol order), then our calculus derives only three clauses and then terminates.

6. Evaluation

We evaluated the performance of ontology classification Sequoia by comparing it with seven state-of-the-art ontology reasoners. We used the evaluation methodology by Steigmiller et al. (2014), which we describe before discussing the evaluation results.

We used the Oxford Ontology Repository[5] as our source of test data. The expressiveness of the ontologies in the repository ranges from simple languages such as DL-Lite and $\mathcal{EL}$ to complex languages such as $\mathcal{SROIQ}(\mathcal{D})$. Each ontology is uniquely identified by an ID. The repository also provides a metadata page that for each ontology lists the ontology language and several measures of ontology size. Seven ontologies in the repository lie outside the OWL 2 DL profile due to irregular RBoxes; since such axioms make reasoning undecidable in general, we simply removed all axioms causing profile violations. Moreover, nine ontologies in the repository could not be parsed by the OWL API due to syntax errors, which we fixed manually. Finally, Sequoia does not support datatypes, nominals, or ABoxes, so we replaced all datatypes and nominals with fresh classes, we replaced data properties with object properties, and we removed all ABox assertions. We thus obtained a test corpus of 703 ontologies, out of which 294 were in $\mathcal{ELH}$.

We tested the latest development version of Sequoia 0.6.1. We used HermiT 1.3.8, Pellet 2.4.0, FaCT++ 1.6.5, and Konclude 0.6.2 for tests over the entire corpus. Also, to verify that our algorithm indeed exhibits a pay-as-you-go behaviour, we also compared our system with ELK 0.4.0, Snorocket 2.8.1, and jcel 0.24.1 on the $\mathcal{ELH}$ ontologies of our test corpus. We used all reasoners in single-threaded mode, apart from Snorocket which does not provide a single-threaded mode (i.e., the system is hard-coded to use all available processors). We were thus able to compare the underlying calculi independently of any orthogonal optimisations such as computation parallelisation. Also, we configured Sequoia to use the cautious strategy. All systems, ontologies, and test results are available online.[6]

We run our experiments on a Dell server with 512 GB of RAM and two Intel CPU E5-2640 V3 2.60 GHz processors with eight cores per processor and two threads per core. The system was running Fedora release 26, kernel version 4.11.9-300.fc26.x86_64, and Java 1.8.0 update 151. We allocated 100 GB of heap memory to each Java reasoner, and a maximum private working set of 100 GB for the native code. In order to prevent individual tests from interfering, for each ontology we started a fresh reasoner process that loaded the ontology using the OWL API and then classified it four times. Each classification task was allowed ten minutes to complete, and we measured its wall-clock time; however, since the Java virtual machine was restarted for each ontology, we discarded the first classification task as warm-up. Thus, if all four classification tasks succeeded, we report the average of

5. http://www.cs.ox.ac.uk/isg/ontologies/
6. http://krr-nas.cs.ox.ac.uk/2017/JAIR-cb/

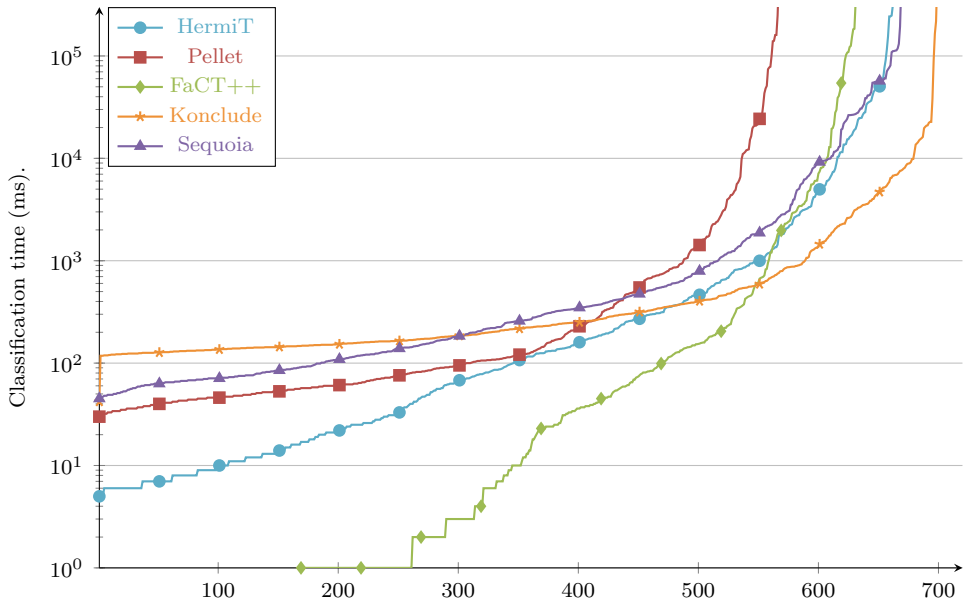


Figure 7: Classification Times for All Ontologies

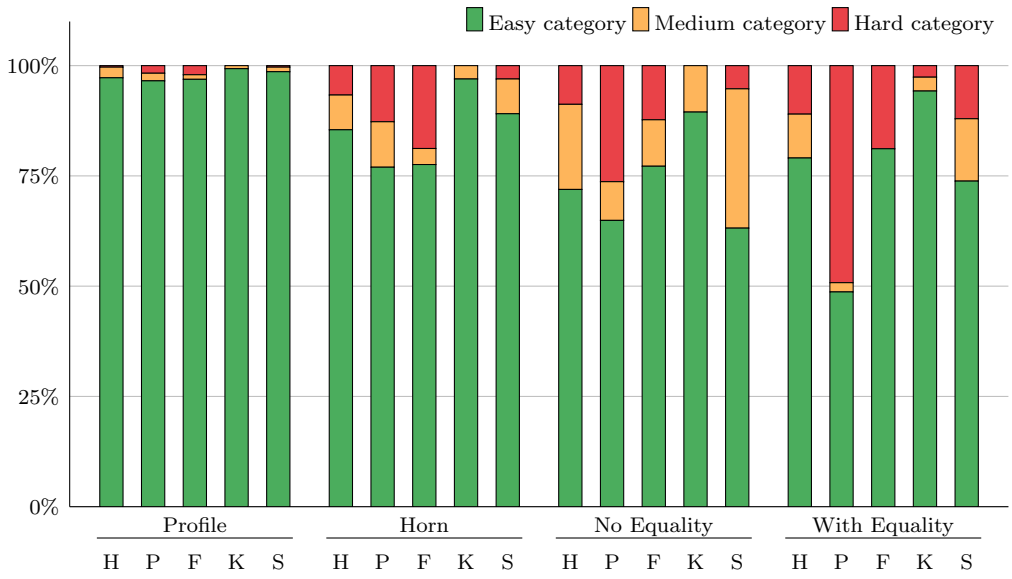


Figure 8: Percentage of Easy, Medium and Hard Ontologies per Ontology Group for HermiT (H), Pellet (P), FaCT++ (F), Konclude (K), and Sequoia (S)

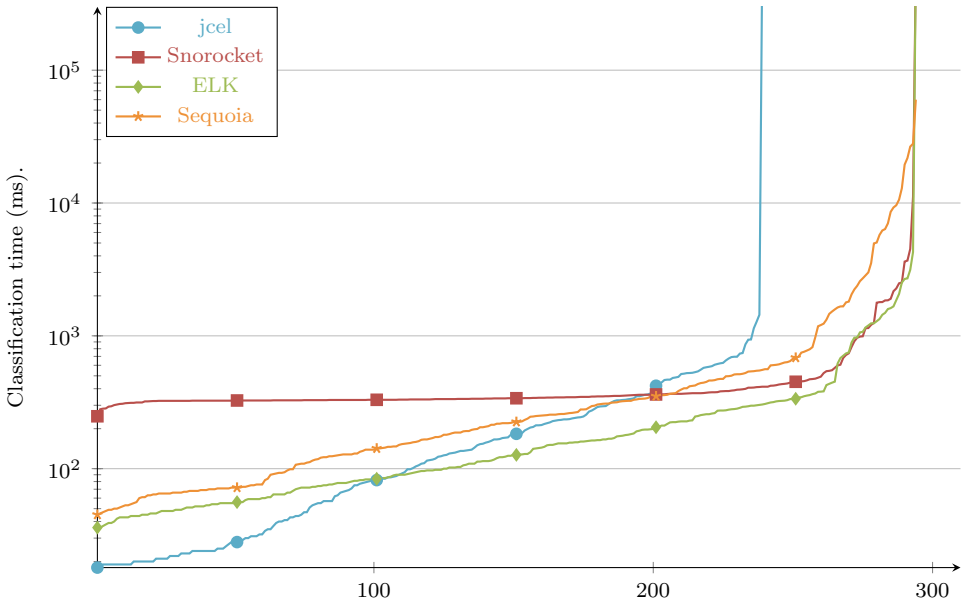


Figure 9: Classification Times for $\mathcal{ELH}$ Ontologies

the last three runs; otherwise, we report the entire test as failed. In all tests we excluded the ontology parsing time in order to analyse the performance of the core reasoning problem.

Figure 7 summarises the classification times for the 703 test ontologies. In particular, for each reasoner, we sorted the average classification times in the ascending order, and we present them by a curve where a point (n, t) indicates that the n-th average classification time from the list is t; the y-axis uses a logarithmic scale and failures are shown as infinity. Figure 9 summarises analogously the classification times for the $\mathcal{ELH}$ ontologies. To further analyse our results, we partitioned all test ontologies into four disjoint groups: in a profile of OWL 2 DL (i.e., in OWL 2 EL, QL, or RL), Horn but not in a profile, disjunctive but without number restrictions (i.e., without equality), and disjunctive and with number restrictions (i.e., with equality). We determined profile membership using the OWL API, and we identified the remaining three groups by analysing the DL-clauses after the clausification step. In addition, for each reasoner, we categorised each ontology as 'easy' ($<$ 10 s), 'medium' (from 10 s to 10 min), or 'hard' (timeout or exception). Figure 8 shows, for each reasoner and ontology group, a bar subdivided into blocks representing the percentage of ontologies in each of the three categories of difficulty.

Sequoia could classify 610 out of 703 ontologies in under 10 s, which is in line with other reasoners. The system was fairly robust and failed only on 32 ontologies; in contrast, HermiT failed on 38, Pellet on 135, FaCT++ on 80, and Konclude on 5 ontologies. Moreover, Sequoia succeeded on 8 ontologies on which HermiT, Pellet, and FaCT++ all failed. Finally, all but four profile ontologies and over 89% of out-of-profile Horn ontologies are easy for Sequoia. Sequoia timed out largely on ontologies containing both disjunctions and equality, and only Konclude was able to process more ontologies of that kind.

Complex number restrictions are the main source of difficulty for Sequoia, as can be seen on the popular Pizza ontology (ID 00799). The ontology contains axioms that link pizza

types with their toppings (e.g., 'Margherita has a mozzarella topping and a tomato toping'), as well as axioms that define broad classes of pizza (e.g., 'A spicy pizza is precisely a pizza that has a spicy toping'). Each instance of the 'pizza' class can be connected with up to 15 objects representing toppings, many of which are introduced via existential quantifiers (e.g., for 'spicy pizza'); however, most of these objects are not necessarily distinct. Finally, the ontology defines an interesting pizza as having at least three toppings, which introduces a DL-clause where four body atoms mention the 'has-topping' relation. Thus, there are 15^4 ways to apply this DL-clause to the clauses defining the toppings, each producing a complex clause that gives rise to numerous other clauses. In contrast, tableaux-based reasoners deal with equalities by guessing an equality relation on the 15 toppings. Most such guesses are consistent since most toppings are not necessarily distinct, so a consistent equality relation can usually be identified quickly. This problem seems inherent in transforming number restrictions as shown in Table 1, and similar behaviour could be observed in the resolution-based reasoner KAON2. To address this issue, one could explore techniques for reasoning with number restrictions using integer linear programming (Haarslev, Timmann, & Möller, 2001) or nondeterministic proof exploration (Kazakov & Klinov, 2014).

On $\mathcal{ELH}$ ontologies, Sequoia, ELK, and Snorocket could classify all ontologies apart from the 'Phenoscape/HierarchyClosure' ontology (ID 00757), which is a very large ontology with more than 1.9 million TBox axioms. In contrast, jcel could not classify 55 ontologies. The performance of Sequoia was similar to that of ELK and Snorocket (where the latter exhibited a fixed initialisation overhead of about 200 ms in each test run), and considerably better than of jcel. Thus, the theoretical pay-as-you-go behaviour described in Proposition 12 can, we believe, actually be observed in practice.

The reasoners' performance diverged substantially on the $\mathcal{ELH}$ version of the Snomed ontology (ID 00798): Sequoia classified this ontology in 131.2 seconds, whereas jcel, ELK, and Snorocket required 56.1, 2.0, and 2.3 seconds, respectively. This significant difference is due to an interaction between Sequoia's indexing strategy and the peculiarities of Snomed. Specifically, Snomed contains several roles (e.g., 'PartOf') that occur in a large number of DL-clauses so, when one of these roles is selected for hyperresolution, Sequoia retrieves a large number of candidate DL-clauses, and for each body atom of each DL-clause it queries the $\mathsf{hyperIndex}_\upsilon$ index for potential hyperresolution candidates, as described in Section 5.3.2. In contrast, other $\mathcal{ELH}$ reasoners specifically optimise this case. For example, ELK indexes context clauses by a pair of a role and a concept (Kazakov et al., 2014), so it can identify the relevant DL-clauses more efficiently. Adapting this technique to Sequoia is not straightforward due to a more complex syntactic form of DL-clauses.

In summary, although Sequoia is an early prototype, it outperforms HermiT, Pellet, and FaCT++ on ontologies whose classification takes at least one second, although Konclude is best performing. Moreover, the system is competitive on $\mathcal{ELH}$ ontologies. Finally, problematic ontologies seem to mostly contain complex RBoxes or large numbers in cardinality restrictions, which suggests promising directions for future optimisation.

7. Conclusion and Future Work

We have presented the first consequence-based calculus for expressive DLs that include both disjunctions and number restrictions. Coupled with well-known preprocessing steps

(Demri & de Nivelle, 2005; Schmidt & Hustadt, 2007; Simančík, 2012), the calculus can handle the DL $\mathcal{SRIQ}$, which covers all of OWL 2 DL apart from nominals, datatypes, and ABox assertions. Our calculus combines ideas from state-of-the-art resolution and (hyper)tableau calculi, such as the use of ordered paramodulation for equality reasoning. Despite its increased complexity, the calculus mimics existing calculi on $\mathcal{EL}$ ontologies. We implemented our calculus in a new DL reasoner called Sequoia. Despite being an early prototype, the system is competitive with several established reasoners.

We are confident that we can extend the calculus to datatypes and ABoxes and thus handle all of OWL 2 DL except nominals. In contrast, handling nominals seems to be much more involved. In fact, adding nominals to $\mathcal{ALCHIQ}^+$ raises the complexity of reasoning to NExpTime (Tobies, 2001) so a worst-case optimal calculus must be nondeterministic, which is quite different from all existing consequence-based calculi we are aware of. Another important challenge is to effectively deal with large numbers in number restrictions.

Acknowledgements

This work was funded by the EPSRC projects DBOnto, MaSI^3, and ED3, and the Royal Society.

Appendix A. Proof of Theorem 8

In this section we prove Theorem 8, which combines two related claims. The first claim states that applying an inference rule to a context structure produces a context structure—that is, all conditions of Definition 6 are satisfied, and in particular all conclusions are context clauses. This is needed since our calculus depends on the syntactic form of context clauses. The second claim is that the resulting context structure is sound—that is, all conclusions are consequences of the ontology. This is shown analogously to the soundness proof of ordered superposition (Nieuwenhuis & Rubio, 1995; Bachmair & Ganzinger, 1998).

Theorem 8 (Soundness)**.** *For an arbitrary expansion strategy, applying an inference rule from Table 2 to an ontology $\mathcal{O}$ and a context structure $\mathcal{D}$ that is sound for $\mathcal{O}$ produces a context structure that is sound for $\mathcal{O}$.*

Proof. Let $\mathcal{O}$ be an ontology and let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathsf{core}, \succcurlyeq, \succ \rangle$ be a context structure that is sound for $\mathcal{O}$. We next show that applying an inference rule from Table 2 to $\mathcal{D}$ using an arbitrary expansion strategy produces a sound context structure. In particular, we show that each derived clause is a context clause according to Definition 1 and satisfies condition S1 of Definition 6. Please note that rules other than Hyper, Eq, and Pred obviously produce context clauses, so we do not stress this any further in our analysis. In addition, the Succ rule can introduce an edge into $\mathcal{D}$, so we show that the edge satisfies condition S2 of Definition 6, and it can refine the order $\succ_v$, so we show that the result is a context term order. Our proof relies on the soundness of hyperresolution: for arbitrary formulas ω, ϕ_i, ψ_i, and γ_i, $1 \leq i \leq n$, with free variables contained in $\vec{x}$, we have

$$\Big\{\forall \vec{x}.\big[\bigwedge_{i=1}^{n} \phi_i \to \omega\big]\Big\} \cup \bigcup_{1 \leq i \leq n} \Big\{\forall \vec{x}.\big[\gamma_i \to \psi_i \vee \phi_i\big]\Big\} \models \forall \vec{x}.\big[\bigwedge_{i=1}^{n} \gamma_i \to \omega \vee \bigvee_{i=1}^{n} \psi_i\big]. \tag{87}$$

We next consider all inference rules from Table 2.

(Core) For each $A \in \mathsf{core}_v$, we clearly have $\mathcal{O} \models \mathsf{core}_v \to A$.

(Hyper) Property (88) clearly holds since $\bigwedge_{i=1}^{n} A_i \to \Delta \in \mathcal{O}$, and property (89) holds since context structure $\mathcal{D}$ is sound for $\mathcal{O}$. But then, property (88) implies property (90), which, together with (87) and (89), implies (91), as required for condition S1.

$$\mathcal{O} \models \bigwedge_{i=1}^{n} A_i \to \Delta \tag{88}$$

$$\mathcal{O} \models \mathsf{core}_v \wedge \Gamma_i \to \Delta_i \vee A_i\sigma \qquad \text{for } 1 \leq i \leq n \tag{89}$$

$$\mathcal{O} \models \bigwedge_{i=1}^{n} A_i\sigma \to \Delta\sigma \tag{90}$$

$$\mathcal{O} \models \mathsf{core}_v \wedge \bigwedge_{i=1}^{n} \Gamma_i \to \Delta\sigma \vee \bigvee_{i=1}^{n} \Delta_i \tag{91}$$

Finally, substitution σ satisfies $\sigma(x) = x$, all premises are context clauses, and $\mathcal{O}$ contains only DL-clauses; thus, the inference rule can only match an atom $S(x, x)$, $S(x, z_i)$ or $S(z_i, x)$ in an ontology clause to atoms $S(y, x)$, $S(x, y)$, $S(x, x)$, $S(f(x), x)$, or $S(x, f(x))$ in the context clause, and so $\sigma(z_i)$ is y, x, or $f(x)$. Consequently, (91) is a context clause.

(Eq) Since $\mathcal{D}$ is sound for $\mathcal{O}$, properties (92) and (93) hold. Now the clause in (94) is a logical consequence of the clauses in (92) and (93), so property (94) holds, as required.

$$\mathcal{O} \models \mathsf{core}_v \wedge \Gamma_1 \to \Delta_1 \vee s_1 \approx t_1 \tag{92}$$

$$\mathcal{O} \models \mathsf{core}_v \wedge \Gamma_2 \to \Delta_2 \vee s_2 \bowtie t_2 \tag{93}$$

$$\mathcal{O} \models \mathsf{core}_v \wedge \Gamma_1 \wedge \Gamma_2 \to \Delta_1 \vee \Delta_2 \vee s_2[t_1]_p \bowtie t_2 \tag{94}$$

Finally, the rule requires that $s_2|_p = s_1$ and that $s_2|_p$ is not a variable, so therefore s_1 is not a variable either. Consequently, term s_1 is always of the form $f(x)$, term t_1 is of the form $g(x)$, x, or y, and term s_2 is of the form $f(x)$, $B(f(x))$, $S(x, f(x))$, or $S(f(x), x)$; thus, $s_2[t_1]_p$ is a context term, and so the result is a context clause.

(Ineq) Since $\mathcal{D}$ is sound for $\mathcal{O}$, we have $\mathcal{O} \models \mathsf{core}_v \wedge \Gamma \to \Delta \vee t \not\approx t$; but then, we clearly have $\mathcal{O} \models \mathsf{core}_v \wedge \Gamma \to \Delta$, as required.

(Factor) Since $\mathcal{D}$ is sound for $\mathcal{O}$, property (95) holds. Moreover, the clause in (96) is a logical consequence of the clause in (95), so property (96) holds, as required.

$$\mathcal{O} \models \mathsf{core}_v \wedge \Gamma \to \Delta \vee s \approx t \vee s \approx t' \tag{95}$$

$$\mathcal{O} \models \mathsf{core}_v \wedge \Gamma \to \Delta \vee t \not\approx t' \vee s \approx t' \tag{96}$$

(Elim) The resulting context structure contains a subset of the clauses from $\mathcal{D}$, so it is clearly sound for $\mathcal{O}$.

(Pred) Since $\mathcal{D}$ is sound for $\mathcal{O}$, properties (97)–(99) hold. Now the clause in (100) is an instance of the clause in (97), so property (100) holds. But then, by (87), properties (98) and (100) imply property (101). Finally, properties (99) and (101) imply property (102).

$$\mathcal{O} \models \mathsf{core}_v \wedge \textstyle\bigwedge_{i=1}^{m} A_i \to \bigvee_{i=m+1}^{m+n} L_i \tag{97}$$

$$\mathcal{O} \models \mathsf{core}_u \wedge \Gamma_i \to \Delta_i \vee A_i\sigma \qquad \text{for } 1 \leq i \leq m \tag{98}$$
$$\mathcal{O} \models \mathsf{core}_u \to \mathsf{core}_v\sigma \tag{99}$$
$$\mathcal{O} \models \mathsf{core}_v\sigma \wedge \textstyle\bigwedge_{i=1}^{m} A_i\sigma \to \bigvee_{i=m+1}^{m+n} L_i\sigma \tag{100}$$
$$\mathcal{O} \models \mathsf{core}_v\sigma \wedge \mathsf{core}_u \wedge \textstyle\bigwedge_{i=1}^{m} \Gamma_i \to \bigvee_{i=1}^{m} \Delta_i \vee \bigvee_{i=m+1}^{m+n} L_i\sigma \tag{101}$$
$$\mathcal{O} \models \mathsf{core}_u \wedge \textstyle\bigwedge_{i=1}^{m} \Gamma_i \to \bigvee_{i=1}^{m} \Delta_i \vee \bigvee_{i=m+1}^{m+n} L_i\sigma \tag{102}$$

For each $m+1 \leq i \leq m+n$, we have $L_i \in \mathsf{Pr}(\mathcal{O})$, so L_i is of the form $B(y)$, $S(x,y)$, $S(y,x)$, or $x \approx y$; but then, $\sigma = \{x \mapsto f(x), y \mapsto x\}$ ensures that $L_i\sigma$ is of the form $B(x)$, $S(f(x),x)$, $S(x,f(x))$, or $f(x) \approx x$ and so $L_i\sigma$ is a context atom, as required.

(Succ) For each clause $A' \to A'$ added to $\mathcal{S}_v$, we clearly have $\mathcal{O} \models \mathsf{core}_v \wedge A' \to A'$, as required for condition S1 of Definition 6. Moreover, assume that the inference rule adds an edge $\langle u, v, f_k \rangle$ to $\mathcal{E}$. Then, the definition of K_1 ensures $\top \to A'\sigma \in \mathcal{S}_u$ for each $A' \in K_1$. Moreover, $\mathcal{D}$ is sound for $\mathcal{O}$, so (103) holds for context u, and it implies (104).

$$\mathcal{O} \models \mathsf{core}_u \to A'\sigma \qquad \text{for each } A' \in K_1 \tag{103}$$
$$\mathcal{O} \models \mathsf{core}_u \to K_1\sigma \tag{104}$$
$$\mathcal{O} \models \mathsf{core}_u \to \mathsf{core}_v\sigma \tag{105}$$

But then, Definition 7 ensures $\mathsf{core}_v \subseteq K_1$, so property (105) holds, as required for condition S2 of Definition 6. Clauses added for atoms in K_2 are tautologies, and therefore they trivially verify the properties. Finally, order $\succ'$ is a context order w.r.t. $\succcurlyeq$ by Definition 7, and so the order of every freshly introduced context is correctly initialised. □

Appendix B. Preliminaries: Rewrite Systems

In the proof of Theorem 9 we construct a model of an ontology, which, as is common in equational theorem proving, we represent using a ground *rewrite system*. To make our proof self-contained, we next recapitulate the definitions of rewrite systems (Nieuwenhuis & Rubio, 1995; Baader & Nipkow, 1998; Bachmair & Ganzinger, 1998). For simplicity, we adapt all standard definitions to ground rewrite systems only.

A (ground) *rewrite system* R is a binary relation on the Herbrand universe HU. Each pair $(s,t) \in R$ is called a *rewrite rule* and is commonly written as $s \Rightarrow t$. The *rewrite relation* $\to_R$ for R is the smallest binary relation on HU such that, for all terms $s_1, s_2, t \in \text{HU}$ and each (not necessarily proper) position p in t, if $s_1 \Rightarrow s_2 \in R$, then $t[s_1]_p \to_R t[s_2]_p$. Moreover, $\overset{*}{\to}_R$ is the reflexive–transitive closure of $\to_R$, and $\overset{*}{\leftrightarrow}_R$ is the reflexive–symmetric–transitive closure of $\to_R$. A term s is *irreducible by* R if no term t exists such that $s \to_R t$; and a literal, clause, or substitution α is *irreducible by* R if each term occurring in α is irreducible by R. Moreover, term t is a *normal form* of s w.r.t. R if $s \overset{*}{\leftrightarrow}_R t$ and t is irreducible by R. We consider the following properties of rewrite systems.

- R is *terminating* if no infinite sequence $s_1, s_2, \ldots$ of terms exists such that, for each i, we have $s_i \to_R s_{i+1}$.
- R is *left-reduced* if, for each $s \Rightarrow t \in R$, the term s is irreducible by $R \setminus \{s \Rightarrow t\}$.

- R is *Church-Rosser* if, for all terms t_1 and t_2 such that $t_1 \overset{*}{\leftrightarrow}_R t_2$, a term z exists such that $t_1 \overset{*}{\rightarrow}_R z$ and $t_2 \overset{*}{\rightarrow}_R z$.

If rewrite system R is terminating and left-reduced, then R is Church-Rosser (Baader & Nipkow, 1998, Theorem 2.1.5 and Exercise 6.7). If R is Church-Rosser, then each term s has a unique normal form t such that $s \overset{*}{\rightarrow}_R t$ holds. The *Herbrand equality interpretation induced by* a rewrite system R is the set R^* such that, for all $s, t \in \mathrm{HU}$, we have $s \approx t \in R^*$ if and only if $s \overset{*}{\leftrightarrow}_R t$.

Term orders can be used to prove termination of rewrite systems. A term order $\succ$ on ground terms (i.e., on HU) is a *simplification order* if the following conditions hold:

1. for all ground terms s_1, s_2, and t, and each position p in t, we have that $s_1 \succ s_2$ implies $t[s_1]_p \succ t[s_2]_p$; and
2. for each term s and each proper position p in s, we have $s \succ s|_p$.

Given a rewrite system R, if a simplification order $\succ$ exists such that $s \Rightarrow t \in R$ implies $s \succ t$, then R is terminating (Baader & Nipkow, 1998, Theorems 5.2.3 and 5.4.8), and, for all ground terms s and t, we have that $s \rightarrow_R t$ implies $s \succ t$.

Appendix C. Proof of Theorem 9

Theorem 9 (Completeness)**.** *Let $\mathcal{O}$ be an ontology, and let $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathsf{core}, \succcurlyeq, \succ \rangle$ be a context structure such that no inference rule from Table 2 is applicable to $\mathcal{O}$ and $\mathcal{D}$. Then, for each query clause $\Gamma_Q \rightarrow \Delta_Q$ and each context $q \in \mathcal{V}$ such that conditions C1 through C3 are satisfied, $\Gamma_Q \rightarrow \Delta_Q \hat{\in} \mathcal{S}_q$ holds.*

C1. $\mathcal{O} \models \Gamma_Q \rightarrow \Delta_Q$.

C2. For each context atom $A \approx \mathsf{true} \in \Delta_Q$ and each context term t such that t does not contain variable y and $A \succ_q t$ holds, we have $t \approx \mathsf{true} \in \Delta_Q$.

C3. For each context atom $A \approx \mathsf{true} \in \Gamma_Q$, we have $\Gamma_Q \rightarrow A \approx \mathsf{true} \hat{\in} \mathcal{S}_q$.

In this section, we fix an ontology $\mathcal{O}$, a context structure $\mathcal{D} = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathsf{core}, \succcurlyeq, \succ \rangle$, a context $q \in \mathcal{V}$, and a query clause $\Gamma_Q \rightarrow \Delta_Q$ such that conditions C2 and C3 of Theorem 9 are satisfied, and we show the contrapositive of condition C1: if $\Gamma_Q \rightarrow \Delta_Q \hat{\notin} \mathcal{S}_q$, then $\mathcal{O} \not\models \Gamma_Q \rightarrow \Delta_Q$. To this end, using a distinguished constant c, the function symbols of sort a, and the symbols of sort p, we construct a Herbrand model that satisfies all clauses in $\mathcal{O}$, but not the query clause $\Gamma_Q \rightarrow \Delta_Q$.

Let t be a term. If t is of the form $t = f(s)$, then s is the *predecessor* of t, and t is a *successor* of s; by these definitions, a constant does not have a predecessor. The a-*neighbourhood* of t is the following set of a-terms: t, $f(t)$ for each $f \in \Sigma^F_{\mathsf{a}}$, and the predecessor t' of t if it exists. The p-*neighbourhood* of t is the following set of p-terms: $B(t)$, $S(t,t)$, $S(t,f(t))$, $S(f(t),t)$, and $B(f(t))$, and, if t has the predecessor t', also $S(t',t)$, $S(t,t')$, and $B(t')$, for each $f \in \Sigma^F_{\mathsf{a}}$, each unary symbol $B \in \Sigma^F_{\mathsf{p}}$, and each binary symbol $S \in \Sigma^F_{\mathsf{p}}$. The *neighbourhood* of t is the set containing the a-neighbourhood of t and the p-neighbourhood

of t. Let σ_t be the substitution such that $\sigma_t(x) = t$; if t has the predecessor t', then $\sigma_t(y) = t'$; and if t is a constant, then $\sigma_t(y) = y$. Finally, we define sets of atoms Pr_t and Su_t as follows:

$$\mathsf{Su}_t = \{\, A\sigma_t \mid A \in \mathsf{Su}(\mathcal{O}) \text{ and } A\sigma_t \text{ is ground} \,\} \tag{106}$$

$$\mathsf{Pr}_t = \{\, L\sigma_t \mid L \in \mathsf{Pr}(\mathcal{O}) \text{ and } L\sigma_t \text{ is ground} \,\} \tag{107}$$

$$\mathsf{Ref}_t = \{\, S(t,t) \mid S \in \Sigma_{\mathsf{p}}^F \text{ is a binary symbol} \,\} \tag{108}$$

C.1 Proof Outline

A key concern in constructing Herbrand equality interpretations is to satisfy the standard properties of equality. For example, if $B(f(g(c)))$ and $g(c) \approx h(c)$ are true, then $B(f(h(c)))$ must be true as well. To address this problem, Herbrand equality interpretations are usually represented using rewrite systems (Nieuwenhuis & Rubio, 1995). For example, given a rewrite system $R = \{B(f(g(c))) \Rightarrow \mathsf{true},\ h(c) \Rightarrow g(c)\}$, atom $B(f(h(c)))$ is 'automatically' satisfied in the induced Herbrand interpretation R^* induced by R.

Thus, we prove Theorem 9 by constructing a rewrite system R so that $R^* \models \mathcal{O}$ and $R^* \not\models \Gamma_Q \to \Delta_Q$ hold. We obtain R by unfolding the context structure $\mathcal{D}$: starting from the context q, we inductively map each a-term t in HU to a context X_t in $\mathcal{D}$, and we use the clauses in $\mathcal{S}_{X_t}$ to construct a *model fragment* R_t—that is, the part of R that satisfies the DL-clauses of $\mathcal{O}$ when x is mapped to t. We obtain R as the union of all R_t. A key issue is to ensure compatibility between adjacent model fragments: when moving from a term t' to its successor $t = f(t')$, combining R_t with $R_{t'}$ should not affect the truth of the DL-clauses of $\mathcal{O}$ at term t'; in other words, the model fragment constructed at t must respect the choices made at t'. We represent these choices by a ground clause $\Gamma_t \to \Delta_t$: conjunction Γ_t contains atoms that are 'inherited' from t' and so must hold at t, and disjunction Δ_t contains atoms that must not hold at t because t' relies on their absence. Construction of the model fragment R_t will ensure

F1. $R_t^* \models N_t$, and

F2. $R_t^* \not\models \Gamma_t \to \Delta_t$—that is, all of Γ_t, but none of Δ_t hold in R_t^*, and so the model fragment at t is compatible with the 'inherited' constraints.

For the induction base, we set $\Gamma_c \to \Delta_c = \Gamma_Q\sigma_c \to \Delta_Q\sigma_c$; this ensures that $R_c^* \not\models \Gamma_c \to \Delta_c$ holds in the initial model fragment, which in turn ensures $R^* \not\models \Gamma_Q \to \Delta_Q$. For the induction step, we shall define $\Gamma_t \to \Delta_t$ based on the model fragment $R_{t'}$ for the predecessor t' of t. To manage the complexity of the proof, we split it into three main parts.

Appendix C.2 deals with the model fragment construction. It takes as input a term t, a context $v = X_t$, and a clause $\Gamma_t \to \Delta_t$. Now let N_t be the set of ground clauses obtained by grounding the context clauses in $\mathcal{S}_v$ as follows.

$$N_t = \{\, \Gamma\sigma_t \to \Delta\sigma_t \mid \Gamma \to \Delta \in \mathcal{S}_v,\ \text{both } \Gamma\sigma_t \text{ and } \Delta\sigma_t \text{ are ground, and } \Gamma\sigma_t \subseteq \Gamma_t \,\} \tag{109}$$

For the construction to work, the following three properties must be satisfied.

L1. $\Gamma_t \to \Delta_t \,\hat{\not\in}\, N_t$.

L2. If $t = c$, then $\Delta_t = \Delta_Q\sigma_c$; and if $t \neq c$, then we have $\{t \approx t'\} \subseteq \Delta_t \subseteq \mathsf{Pr}_t$ where t' is the predecessor of t.

L3. For each $A \in \Gamma_t$, we have $\Gamma_t \to A \mathrel{\hat{\in}} N_t$.

We construct a rewrite system R_t satisfying conditions F1 and F2 by adapting the techniques from paramodulation-based theorem proving. First, we order all clauses in N_t into a sequence $C^i = \Gamma^i \to \Delta^i \vee L^i$, $1 \leq i \leq n$, that is compatible with the context term order $\succ_v$ in a specific way. Next, we initialise R_t to $\emptyset$, and then we examine each clause C^i in this sequence; if C^i does not hold in the model constructed thus far, we make the clause true by adding L^i to R_t. To prove condition F1, we assume for the sake of a contradiction that a clause C^i with smallest i exists such that $R_t^* \not\models C^i$, and we show that an application of the Eq, Ineq, or Factor rule to C^i necessarily produces a clause C^j such that $R_t^* \not\models C^j$ and $j < i$. Conditions L1 through L3 allow us to satisfy condition F2. Due to condition L2 and condition 5 of Definition 3, we can order the clauses in the sequence such that each clause C^i capable of producing an atom from Δ_t comes before any other clause in the sequence; and then we use condition L1 to show that no such clause actually exists. Moreover, condition L3 ensures that all atoms in Γ_t are actually produced in R_t^*.

Appendix C.3 deals with the inductive unfolding of $\mathcal{D}$. We apply the model fragment construction at each step, and a key issue is to show that conditions L1 through L3 are satisfied, which in turn ensures that individual model fragments can safely be combined. For the base case, we map constant c to context $X_c = q$, and we define $\Gamma_c = \Gamma_Q$ and $\Delta_c = \Delta_Q$; then, conditions L1 and L2 hold by definition, and condition L3 holds by condition C3 of Theorem 9. For the induction step, we assume that we have already mapped some term t' to a context $u = X_{t'}$, and we consider the term $t = f(t')$ for each $f \in \Sigma_{\mathsf{a}}^F$.

- If t does not occur in an atom in $R_{t'}$, we let $R_t = \{t \Rightarrow c\}$ and thus make t equal to c. Term t is thus interpreted in exactly the same way as c, so we stop the unfolding.
- If $R_{t'}$ contains a rule $t \Rightarrow s$, then t and s are equal, and so we interpret t exactly as s; hence, we stop the unfolding.
- In all other cases, the Succ rule ensures that $\mathcal{D}$ contains an edge $\langle u, v, f \rangle$ such that v satisfies all preconditions of the rule, so we define $X_t = v$. Furthermore, we let $\Gamma_t = R_{t'}^* \cap \mathsf{Su}_t$ be the set of atoms that hold at t' and are relevant to t, and we let $\Delta_t = \mathsf{Pr}_t \setminus R_{t'}^*$ be the set of atoms that do not hold at t' and are relevant to t. We finally show that such Γ_t and Δ_t satisfy condition L1: if this were not the case, the Pred rule would derive a clause in $N_{t'}$ that is not true in $R_{t'}^*$.

We finally define R as the union of all R_t from the above construction. We also prove that R is a Church-Rosser rewrite system, so each term has a unique normal form w.r.t. R.

Appendix C.4 completes the proof of Theorem 9. To prove $R^* \models \mathcal{O}$, we consider a DL-clause $\Gamma \to \Delta \in \mathcal{O}$ and a substitution τ that makes the clause ground. Since R is Church-Rosser, we can consider only τ that are irreducible by R—that is, that do not contain terms that can we rewritten using the rules in R. Since each model fragment satisfies condition F2, we can evaluate $\Gamma\tau \to \Delta\tau$ in $R_{\tau(x)}^*$ instead of R^*. Moreover, we show that $R_{\tau(x)}^* \models \Gamma\tau \to \Delta\tau$ holds: if that were not the case, the Hyper rule would derive a clause in $N_{\tau(x)}$ that violates condition F1. Finally, we analogously show that $R^* \not\models \Gamma_Q \to \Delta_Q$ holds as well, which completes our proof.

C.2 Constructing a Model Fragment

In this section, we show how, given a term t, we can generate a fragment of the model of $\mathcal{O}$ that covers the neighbourhood of t. In the rest of Appendix C.2, we fix the following parameters for the model fragment generation process:

- t is a ground a-term,
- v is a context in $\mathcal{D}$,
- Γ_t is a conjunction of atoms, and
- Δ_t is a disjunction of atoms.

Let N_t be the as defined in (109), and assume that conditions L1 through L3 hold. We next construct a rewrite system R_t such that $R_t^* \models N_t$ and $R_t^* \not\models \Gamma_t \rightarrow \Delta_t$ holds.

Throughout Appendix C.2, we treat the terms in the a-neighbourhood of t as if they were independent constants. Thus, even though the rewrite system R_t will contain terms t and $f(t)$, we will not consider terms with further nesting. This simplifies the formal treatment in this section as we do not need to worry about how the predecessors or the successors of t are ordered (e.g., in the proof of Lemma 13), or the possibility that the term t inside a term $f(t)$ might be reduced by a rewrite rule. Note, however, that conditions 1 through 4 of Definition 3 impose a global order on all context terms, which in turn imposes a consistent global ordering on all ground a-terms. Moreover, when unfolding the context structure in Appendix C.3, condition M2 ensures that no subterm of $f(t)$ is ever reduced. Thus, we can still connect all model fragments into one Church-Rosser rewrite system (see Lemma 26).

C.2.1 Grounding the Context Order

To construct R_t, we need an order on the terms in the neighbourhood of t that is compatible with $\succ_v$. To this end, let $>_t$ be a total, strict, simplification order on the set of ground terms constructed using the terms in the a-neighbourhood of t (which we treat as independent constants without any subterms) and the symbols of the predicate sort p; for all context terms s_1 and s_2 such that $s_1\sigma_t$ and $s_2\sigma_t$ are both ground, and for t' the predecessor of t if it exists, order $>_t$ must satisfy the following conditions.

O1. $s_1 \succ_v s_2$ implies $s_1\sigma_t >_t s_2\sigma_t$.

O2. $s_1\sigma_t \approx \mathsf{true} \in \Delta_t$ and $s_2\sigma_t \notin \{t, t', \mathsf{true}\}$ and $s_2\sigma_t \approx \mathsf{true} \notin \Delta_t$ imply $s_2\sigma_t >_t s_1\sigma_t$.

Lemma 13. *There exists at least one order $>_t$ that satisfies conditions O1 and O2.*

Proof. Note that condition O2 can be equivalently rewritten as follows ($*$):

$$s_1\sigma_t \approx \mathsf{true} \in \Delta_t \text{ and } s_2\sigma_t \notin \{t, t', \mathsf{true}\} \text{ and } s_1\sigma_t \geq_t s_2\sigma_t \text{ imply } s_2\sigma_t \approx \mathsf{true} \in \Delta_t.$$

We can construct $>_t$ in several steps. First, for all context terms s_1 and s_2 such that $s_1 \succ_v s_2$ and $s_1\sigma_t$ and $s_2\sigma_t$ are both ground, we define $s_1\sigma_t >_t s_2\sigma_t$. The result satisfies condition O1. Now let $M = \{\, A \mid A \approx \mathsf{true} \in \Delta_t \,\}$ and consider the following two cases.

- Assume $t = c$. For arbitrary terms $s_1\sigma_t \in M$ and $s_2\sigma_t \notin \{c, \mathsf{true}\}$ with $s_1\sigma_t \geq_t s_2\sigma_t$, term s_2 does not contain y (since Δ_Q contains only variable x); moreover, $s_1\sigma_t \in M$ implies $s_1 \approx \mathsf{true} \in \Delta_Q$, so condition C2 of Theorem 9 ensures $s_2 \approx \mathsf{true} \in \Delta_Q$; hence, we have $s_2\sigma_t \in M$, as required for $(*)$. Furthermore, Δ_Q is a query clause, so all terms in M are of the form $B(c)$ or $S(c, c)$; thus, due to conditions 1 and 4 of Definition 3, each $A \in M$ can be $\geq_t$-larger only than c and true. Thus, we can extend $>_t$ by totally ordering all terms of M and placing them immediately after c and true in the ordering. The result clearly satisfies conditions O1 and O2.

- Assume $t \neq c$. For arbitrary terms $s_1\sigma_t \in M$ and $s_2\sigma_t \notin \{t, t', \mathsf{true}\}$, condition $\Delta_t \subseteq \mathsf{Pr}_t$ ensures that all terms in M are of the form $B(t')$, $S(t, t')$, or $S(t', t)$ and moreover $s_1 \approx \mathsf{true} \in \mathsf{Pr}(\mathcal{O})$; in addition, $s_2\sigma_t \notin \{t, t', \mathsf{true}\}$ ensures $s_2 \notin \{x, y, \mathsf{true}\}$; but then, condition 5 of Definition 3 ensures $s_1 \not\succ_v s_2$, so $s_1\sigma_t \not\succ_t s_2\sigma_t$ by the order $>_t$ constructed thus far; thus, the only possibility is $s_1\sigma_t = s_2\sigma_t$, in which case $(*)$ holds trivially. Furthermore, due to condition 1 and condition 4, each $A \in M$ can be $\geq_t$-larger only than t, t', and true. Therefore, we can extend $>_t$ by totally ordering all terms of M and placing them immediately after t, t', and true in the ordering. The result clearly satisfies conditions O1 and O2. Note that in this section we treat all a-terms as constants, so we do not need to worry about defining the order on the predecessor of t' or on the ancestors of $f(t)$.

Finally, we can arbitrarily extend $\geq_t$ to a total order, and the result also satisfies conditions O1 and O2. □

We can choose any such $>_t$, and we extend it to ground literals (also written $>_t$) by associating each $s \not\approx t$ with the multiset $\{s, s, t, t\}$ and each $s \approx t$ with the multiset $\{s, t\}$, and then comparing the result using the multiset extension of the term order (as defined in Section 2). Due to condition 1 of Definition 3, we have

$$A \approx \mathsf{true} >_t t \not\approx t >_t t \approx t >_t t \not\approx t' >_t t \approx t' >_t t' \not\approx t' >_t t' \approx t' >_t \mathsf{true} \approx \mathsf{true} \qquad (110)$$

for each context atom A of the form $B(t)$, $B(t')$, $S(t, t')$, $S(t', t)$, and $S(t, t)$. Finally, we further extend $>_t$ to disjunctions of ground literals (also written $>_t$) by identifying each disjunction $\bigvee_{i=1}^{n} L_i$ with the multiset $\{L_1, \ldots, L_n\}$ and then comparing the result using the multiset extension of the literal order.

C.2.2 Constructing the Rewrite System R_t

In resolution theorem proving, the head and the body of a clause are usually multisets of literals (Bachmair & Ganzinger, 2001). However, we treat the head and body as sets (i.e., no duplication is possible). This simplifies the formal treatment without loss of generality: clause $\Gamma \to \Delta \vee L \vee L$ (resp. $\Gamma \wedge A \wedge A \to \Delta$) entails $\Gamma \to \Delta \vee L$ (resp. $\Gamma \wedge A \to \Delta$), and the latter clause is always strictly smaller than the former in the clause order.

We arrange all clauses in N_t into a sequence $C^1, \ldots, C^n$. Since the body of each C^i is a subset of Γ_t, no C^i can have an empty head since that would contradict condition L1. Moreover, the heads of the clauses in N_t do not contain repetitions. Consequently, we can assume that each C^i is of the form $C^i = \Gamma^i \to \Delta^i \vee L^i$ where $L^i >_t \Delta^i$, literal L^i is of the

form $L^i = l^i \bowtie r^i$ with $\bowtie \in \{\approx, \not\approx\}$, and $l^i \geq_t r^i$. For the rest of Appendix C.2, we reserve C^i, Γ^i, Δ^i, L^i, l^i, and r^i for referring to the (parts of) the clauses in this sequence. Finally, without loss of generality we assume that, for all $1 \leq i < j \leq n$, we have $\Delta^j \vee L^j \geq_t \Delta^i \vee L^i$.

We next define the sequence $R_t^0, \ldots, R_t^n$ of rewrite systems by setting $R_t^0 = \emptyset$ and defining each R_t^i with $1 \leq i \leq n$ inductively as follows:

- $R_t^i = R_t^{i-1} \cup \{l^i \Rightarrow r^i\}$ if L^i is of the form $l^i \approx r^i$ such that
  - R1. $(R_t^{i-1})^* \not\models \Delta^i \vee l^i \approx r^i$,
  - R2. $l^i >_t r^i$,
  - R3. l^i is irreducible by R_t^{i-1}, and
  - R4. $(R_t^{i-1})^* \not\models s \approx r^i$ for each $l^i \approx s \in \Delta^i$; and
- $R_t^i = R_t^{i-1}$ in all other cases.

Finally, let $R_t = R_t^n$; we call R_t the *model fragment for t, v, Γ_t, and Δ_t*. Each clause $C^i = \Gamma^i \to \Delta^i \vee l^i \approx r^i$ that satisfies the first condition in the above construction is called *generative*, and the clause is said to *generate* the rule $l^i \Rightarrow r^i$ in R_t.

C.2.3 The Properties of the Model Fragment R_t

Lemma 14. *The rewrite system R_t is Church-Rosser.*

Proof. To see that R_t is terminating, simply note that, for each rule $l \Rightarrow r \in R_t$, condition R2 ensures $l >_t r$, and that $>_t$ is a simplification order.

To see that R_t is left-reduced, consider an arbitrary rule $l^i \Rightarrow r^i \in R_t$ that is added to R_t in step i of the clause sequence. By condition R3, term l^i is irreducible by R_t^{i-1}. Now consider an arbitrary rule $l^j \Rightarrow r^j \in R_t$ that is added to R_t at any step j of the construction where $j > i$. The definition of the clause order implies $l^j \approx r^j \geq_t l^i \approx r^i$; since $l^j >_t r^j$ and $l^i >_t r^i$ by condition R2, by the definition of the literal order we have $l^j \geq_t l^i$. Since $l^i \Rightarrow r^i \in R_t^{j-1}$, condition R3 for j ensures $l^i \neq l^j$, and so we have $l^j >_t l^i$; consequently, l^j is not a subterm of l^i, and thus term l^i is irreducible by $R_t^j \setminus \{l^i \Rightarrow r^i\}$. □

Lemma 15. *For each $1 \leq i \leq n$ and each $l \not\approx r \in \Delta^i \vee L^i$, we have $(R_t^{i-1})^* \models l \approx r$ if and only if $R_t^* \models l \approx r$.*

Proof. Consider an arbitrary clause $C^i = \Gamma^i \to \Delta^i \vee L^i$ and an arbitrary inequality $l \not\approx r$ such that $l \not\approx r \in \Delta^i \vee L^i$. If $(R_t^{i-1})^* \models l \approx r$, then $R_t^{i-1} \subseteq R_t$ implies $R_t^* \models l \approx r$, as required. Now assume that $(R_t^{i-1})^* \not\models l \approx r$. Let l' and r' be the normal forms of l and r, respectively, w.r.t. R_t^{i-1}. Now consider an arbitrary j with $i \leq j \leq n$ such that $l^j \Rightarrow r^j$ is generated by C^j. The definitions of the model construction ensure $l^j \approx r^j \geq_t L^i >_t \Delta^i$. Now if $l \not\approx r = L^i$, then $l^j \approx r^j >_t l \not\approx r$ holds because the two literals are different; otherwise, we have $l \not\approx r \in \Delta^i$, and so $l^j \approx r^j >_t l \not\approx r$ holds as well. By the definition of the literal order, we then have $l^j >_t l \geq_t l'$ and $l^j >_t r \geq_t r'$; since $>_t$ is a simplification order, l^j is a subterm of neither l' nor r'. Thus, l' and r' are the normal forms of l and r, respectively, w.r.t. R_t^j, and so we have $(R_t^j)^* \not\models l' \approx r'$; but then, we have $(R_t^j)^* \not\models l \approx r$, as required. □

Lemma 16. *For each generative clause $\Gamma^i \to \Delta^i \vee l^i \approx r^i$, we have $R_t^* \not\models \Delta^i$.*

Proof. Consider an arbitrary generative clause $C^i = \Gamma^i \to \Delta^i \vee l^i \approx r^i$ and an arbitrary literal $L \in \Delta^i$; condition R1 ensures that $(R_t^{i-1})^* \not\models L$. We next show that $R_t^* \not\models L$.

Assume that L is of the form $l \not\approx r$. Since $l \not\approx r \in \Delta^i \vee l^i \approx r^i$, by Lemma 15 we have $R_t^* \not\models L$, as required.

Assume that L is of the form $l \approx r$ with $l >_t r$. We show by induction that, for each j with $i \leq j \leq n$, we have $(R_t^j)^* \not\models L$. To this end, we assume that $(R_t^{j-1})^* \not\models L$. If C^j is not generative, then $R_t^j = R_t^{j-1}$, and so $(R_t^j)^* \not\models L$. The only remaining possibility is that C^j is generative, for which we consider the following cases.

- $l^j = l$. We have the following two subcases.
  - $j = i$, and so $l = l^i = l^j$. Condition R4 ensures $(R_t^{i-1})^* \not\models r \approx r^i$. Let r' and r'' be the normal forms of r and r^i, respectively, w.r.t. R_t^{i-1}; we have $(R_t^{i-1})^* \not\models r' \approx r''$. Moreover, $l >_t r \geq_t r'$ and $l >_t r^i \geq_t r''$ hold; since $>_t$ is a simplification order, l is a subterm of neither r' nor r''; therefore, r' and r'' are the normal forms of r and r^i, respectively, w.r.t. R_t^i, and therefore $(R_t^i)^* \not\models r' \approx r''$. Finally, since $l \Rightarrow r^i \in R_t^i$, term r'' is the normal form of l w.r.t. R_t^i, and so $(R_t^i)^* \not\models l \approx r$.
  - $j > i$. But then, $l^j \approx r^j \geq_t l^i \approx r^i >_t l \approx r$ implies $l^j = l^i = l$. Furthermore, C^i is generative, so we have $l^i \Rightarrow r^i \in R_t^{j-1}$. But then, l^j is not irreducible by R_t^{j-1}, which contradicts condition R3.
- $l^j >_t l$. Let l' and r' be the normal forms of l and r, respectively, w.r.t. R_t^{j-1}. Then, we have $l^j >_t l \geq_t l'$ and $l^j >_t r \geq_t r'$; since $>_t$ is a simplification order, l^j is a subterm of neither l' nor r'. Thus, l' and r' are the normal forms of l and r, respectively, w.r.t. R_t^j, and so $(R_t^j)^* \not\models l' \approx r'$; hence, $R_t^* \not\models l \approx r$ holds. □

Lemma 17. *Let $\Gamma \to \Delta$ be a clause such that $\Gamma \to \Delta \,\hat{\in}\, N_t$. Then $R_t^* \models \Delta$ holds if there exists i with $1 \leq i \leq n+1$ such that*

1. *for each $1 \leq j < i$, we have $R_t^* \models \Delta^j \vee L^j$, and*
2. *if $i \leq n$ (i.e., i is an index of a clause from N_t), then $\Delta^i \vee L^i >_t \Delta$.*

Proof. Assume that $\Gamma \to \Delta \,\hat{\in}\, N_t$ holds. If $\Gamma \to \Delta$ satisfies condition 1 of Definition 4, then we clearly have $R_t^* \models \Delta$. Assume that $\Gamma \to \Delta$ satisfies condition 2 of Definition 4 due to some clause $\Gamma^j \to \Delta^j \vee L^j \in N_t$ such that $\Gamma^j \subseteq \Gamma$ and $\Delta^j \cup \{L^j\} \subseteq \Delta$ hold; the latter clearly implies $\Delta \geq_t \Delta^j \vee L^j$. Let i be an integer satisfying this lemma's assumption. If $i = n+1$, then we clearly have $j < i$; otherwise, $\Delta^i \vee L^i >_t \Delta$ implies $\Delta^i \vee L^i >_t \Delta^j \vee L^j$, and so we also have $j < i$. But then, by the lemma assumption we have $R_t^* \models \Delta^j \vee L^j$, which implies $R_t^* \models \Delta$, as required. □

Lemma 18. *For each clause $\Gamma \to \Delta$ such that $\Gamma\sigma_t$ and $\Delta\sigma_t$ are ground and $\Gamma \to \Delta \,\hat{\in}\, \mathcal{S}_v$ and $\Gamma\sigma_t \subseteq \Gamma_t$ both hold, we have $\Gamma\sigma_t \to \Delta\sigma_t \,\hat{\in}\, N_t$.*

Proof. Assume that $\Gamma \to \Delta \,\hat{\in}\, \mathcal{S}_v$ holds. If $\Gamma \to \Delta$ satisfies condition 1 of Definition 4, then terms s and s' exist such that $s \approx s \in \Delta$ or $\{s \approx s',\ s \not\approx s'\} \subseteq \Delta$; but then, $s\sigma_t \approx s\sigma_t \in \Delta\sigma_t$ or $\{s\sigma_t \approx s'\sigma_t,\ s\sigma_t \not\approx s'\sigma_t\} \subseteq \Delta\sigma_t$, so $\Gamma\sigma_t \to \Delta\sigma_t \,\hat{\in}\, N_t$ holds. Furthermore, if $\Gamma \to \Delta$ satisfies condition 2 of Definition 4, then clause $\Gamma' \to \Delta' \in \mathcal{S}_v$ exists such that $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$; but then, since $\Gamma'\sigma_t \subseteq \Gamma\sigma_t \subseteq \Gamma_t$ holds and moreover both $\Gamma\sigma_t$ and $\Delta\sigma_t$ are ground, we have that $\Gamma'\sigma_t \to \Delta'\sigma_t \in N_t$ holds, and so $\Gamma\sigma_t \to \Delta\sigma_t \,\hat{\in}\, N_t$ holds as well. □

Lemma 19. *For each clause $\Gamma' \to \Delta' \vee s' \not\approx s' \in N_t$, we have $\Gamma' \to \Delta' \,\hat{\in}\, N_t$.*

Proof. Consider an arbitrary clause $\Gamma' \to \Delta' \vee s' \not\approx s' \in N_t$. By the definition of N_t, a clause $\Gamma \to \Delta \vee s \not\approx s \in \mathcal{S}_v$ exists such that

$$\Gamma\sigma_t = \Gamma' \subseteq \Gamma_t, \quad \Delta\sigma_t = \Delta', \quad \text{and} \quad s\sigma_t = s'. \tag{111}$$

By the assumption of Theorem 9, the Ineq rule is not applicable to $\Gamma \to \Delta \vee s \not\approx s$, and so we have $\Gamma \to \Delta \,\hat{\in}\, \mathcal{S}_v$. Since $\Gamma\sigma_t \subseteq \Gamma_t$, by Lemma 18 we have $\Gamma' \to \Delta' \,\hat{\in}\, N_t$, as required. □

Lemma 20. *Both t and t' (if it exists) are irreducible by R_t.*

Proof. Recall that throughout Appendix C.2, both t and t' are treated as constants, and that each rule $l \Rightarrow r \in R_t$ satisfies $l >_t r$ due to condition R2. Now if $t = c$, then t' does not exist and t is the smallest a-term in $>_t$ and therefore it cannot occur on the left-hand side of a rule in R_t, and so the lemma holds; hence, in the rest of this proof we assume that $t \neq c$. Condition L2 then ensures $t \approx t' \in \Delta_t$. Moreover, condition 1 of Definition 3 and condition O1 imply $t >_t t'$, and $t' \not>_t s$ for each a-term s. Therefore, t' is irreducible; moreover, to show that t is irreducible, we next prove that $t \Rightarrow t' \notin R_t$.

For the sake of a contradiction, assume that a clause $\Gamma \to \Delta \vee t \approx t' \in N_t$ exists that generates the rule $t \Rightarrow t'$ in R_t. By the definition of N_t, we have $\Gamma \subseteq \Gamma_t$. Since $t \approx t' >_t \Delta$, we have $\Delta \subseteq \{t' \not\approx t', t' \approx t'\}$. Since the clause is generative, condition R1 ensures $t' \approx t' \notin \Delta$, and so we have $\Delta \subseteq \{t' \not\approx t'\}$. If $\Delta \neq \emptyset$, then Lemma 19 implies $\Gamma \to t \approx t' \,\hat{\in}\, N_t$, and otherwise $\Gamma \to t \approx t' \,\hat{\in}\, N_t$ holds by our assumption. But then, together with $t \approx t' \in \Delta_t$, this implies $\Gamma_t \to \Delta_t \,\hat{\in}\, N_t$, which contradicts condition L1. □

Lemma 21. *For each $\Gamma \to \Delta \in N_t$, we have $R_t^* \models \Delta$.*

Proof. For the sake of a contradiction, choose $C^i = \Gamma^i \to \Delta^i \vee L^i$ as the clause in the sequence of clauses from Appendix C.2.2 with the smallest i such that $R_t^* \not\models \Delta^i \vee L^i$; recall that $L^i >_t \Delta^i$ and $L^i = l^i \bowtie r^i$ with $\bowtie \in \{\approx, \not\approx\}$. Due to our choice of i, condition (1) of Lemma 17 holds for C^i and i. By the definition of N_t, a clause $\Gamma \to \Delta \vee L \in \mathcal{S}_v$ exists where

$$\Gamma\sigma_t = \Gamma^i \subseteq \Gamma_t, \quad \Delta\sigma_t = \Delta^i, \quad L\sigma_t = L^i, \quad \text{and} \quad \Delta \not\succeq_v L. \tag{112}$$

We next prove the claim of this lemma by considering the possible forms of L^i.

Assume $L^i = l^i \approx r^i$ with $l^i = r^i$. But then, we have $R_t^* \models L^i$, which contradicts our assumption that $R_t^* \not\models \Delta^i \vee L^i$.

Assume $L^i = l^i \approx r^i$ with $l^i >_t r^i$. But then, literal L is of the form $l \approx r$ and it satisfies $l\sigma_t \approx r\sigma_t = l^i \approx r^i$. By the definition of $>_t$, we have $l \succ_v r$. We first show that $(R_t^{i-1})^* \not\models \Delta^i \vee L^i$ holds; towards this goal, note that, for each equality $s_1 \approx s_2 \in \Delta^i \vee L^i$, properties $R_t^* \not\models s_1 \approx s_2$ and $R_t^{i-1} \subseteq R_t$ imply $(R_t^{i-1})^* \not\models s_1 \approx s_2$; and for each inequality $s_1 \not\approx s_2 \in \Delta^i$, Lemma 15 and $R_t^* \not\models s_1 \not\approx s_2$ imply $(R_t^{i-1})^* \not\models s_1 \not\approx s_2$. Thus, clause C^i satisfies conditions R1 and R2; however, since $R_t^* \not\models l^i \approx r^i$, clause C^i is not generative and thus either condition R3 or condition R4 is not satisfied. We next consider both possibilities.

- Condition R3 does not hold—that is, l^i is reducible by R_t^{i-1}. By the definition of reducibility, a position p and a clause $C^j = \Gamma^j \to \Delta^j \vee l^j \approx r^j$ generating the rule $l^j \Rightarrow r^j$ exist such that $j < i$ and $l^i|_p = l^j$. By Lemma 20, l^j is neither t nor t', and therefore $l|_p$ is neither x nor y. Due to $j < i$, we have $l^i \approx r^i \geq_t l^j \approx r^j$; together with $l^j \approx r^j >_t \Delta^j$, we have that $l^i \approx r^i >_t \Delta^j$. Lemma 16 ensures $R_t^* \not\models \Delta^j$, and the definition of N_t ensures that a clause $\Gamma' \to \Delta' \vee l' \approx r' \in \mathcal{S}_v$ exists such that

$$\Gamma'\sigma_t = \Gamma^j \subseteq \Gamma_t, \;\; \Delta'\sigma_t = \Delta^j, \;\; l'\sigma_t = l^j, \;\; r'\sigma_t = r^j, \;\; \Delta' \not\succeq_v l' \approx r', \;\; \text{and} \;\; l' \succ_v r'. \tag{113}$$

  By the assumption of Theorem 9, the Eq rule is not applicable to (112) and (113), and so $\Gamma \wedge \Gamma' \to \Delta \vee \Delta' \vee l[r']_p \approx r \;\hat{\in}\; \mathcal{S}_v$. Let $\Delta'' = \Delta^i \vee \Delta^j \vee l^i[r^j]_p \approx r^i$. Then clearly $\Gamma\sigma_t \cup \Gamma'\sigma_t \subseteq \Gamma_t$, so Lemma 18 ensures that $\Gamma^i \wedge \Gamma^j \to \Delta'' \;\hat{\in}\; N_t$ holds. Set R_t^* is a congruence, so $l^i[r^j]_p \approx r^i \notin R_t^*$ holds, and therefore $R_t^* \not\models \Delta''$ holds. Finally, $>_t$ is a simplification order, which ensures $l^i \approx r^i >_t l^i[r^j]_p \approx r^i$; together with $l^i \approx r^i >_t \Delta^i$ and $l^i \approx r^i >_t \Delta^j$, we have $l^i \approx r^i >_t \Delta''$. But then, Lemma 17 implies $R_t^* \models \Delta''$, which is a contradiction.

- Condition R4 does not hold. Then, some term s exists such that $l^i \approx s \in \Delta^i$ and $(R_t^{i-1})^* \models s \approx r^i$. Due to $R_t^{i-1} \subseteq R_t$, we have $R_t^* \models s \approx r^i$, and so $R_t^* \not\models s \not\approx r^i$. Furthermore, $\Delta \vee L$ is of the form $\Delta' \vee l \approx r \vee l' \approx r'$ such that

$$l\sigma_t = l^i, \quad r\sigma_t = s, \quad l'\sigma_t = l^i, \quad \text{and} \quad r'\sigma_t = r^i. \tag{114}$$

  We then have $l' = l$. By the assumption of Theorem 9, the Factor rule is not applicable to $\Gamma \to \Delta \vee L$, so $\Gamma \to \Delta' \vee r \not\approx r' \vee l' \approx r' \;\hat{\in}\; \mathcal{S}_v$. Let $\Delta'' = \Delta'\sigma_t \vee s \not\approx r^i \vee l^i \approx r^i$. But then, $\Gamma\sigma_t \subseteq \Gamma_t$ and Lemma 18 ensure that $\Gamma^i \to \Delta'' \;\hat{\in}\; N_t$ holds. Now $\Delta' \subseteq \Delta$ implies $\Delta'\sigma_t \subseteq \Delta^i$, so we have $R_t^* \not\models \Delta'\sigma_t$, and therefore $R_t^* \not\models \Delta''$. Moreover, $l^i >_t r^i$ and $l^i >_t s$ imply $l^i \approx r^i >_t s \approx r^i$; thus, $\Delta^i \vee l^i \approx r^i >_t \Delta''$ holds. Lemma 17 then implies $R_t^* \models \Delta''$, which is a contradiction.

Assume $L^i = l^i \not\approx r^i$ with $l^i = r^i$. Then, literal L is of the form $l \not\approx r$ and it satisfies $l\sigma_t \not\approx r\sigma_t = l^i \not\approx r^i$. But then, $l^i = r^i$ implies $l = r$. By Lemma 19, we have $\Gamma^i \to \Delta^i \;\hat{\in}\; N_t$. Clearly, $\Delta^i \vee l^i \not\approx r^i >_t \Delta^i$, and so Lemma 17 implies $R_t^* \models \Delta^i$, which is a contradiction.

Assume $L^i = l^i \not\approx r^i$ with $l^i >_t r^i$. Lemma 15 ensures $(R_t^{i-1})^* \not\models l^i \not\approx r^i$; hence, l^i is reducible by R_t^{i-1} so, by the definition of reducibility, a position p and a generative clause $C^j = \Gamma^j \to \Delta^j \vee l^j \approx r^j$ exist such that $j < i$ and $l^i|_p = l^j$. Due to $j < i$, we have that $l^i \not\approx r^i >_t l^j \approx r^j >_t \Delta^j$. Lemma 16 ensures $R_t^* \not\models \Delta^j$, and the definition of N_t ensures that a clause $\Gamma' \to \Delta' \vee l' \approx r' \in \mathcal{S}_v$ exists satisfying (113), as in the first case. Now if term l^j were either t or t', this would contradict Lemma 20; hence, $l|_p$ is neither x nor y. By the assumption of Theorem 9, the Eq rule is not applicable to clauses (112) and (113), and so $\Gamma \wedge \Gamma' \to \Delta \vee \Delta' \vee l[r']_p \not\approx r \;\hat{\in}\; \mathcal{S}_v$ holds. Let $\Delta'' = \Delta^i \vee \Delta^j \vee l^i[r^j]_p \not\approx r^i$. We clearly have $\Gamma\sigma_t \cup \Gamma'\sigma_t \subseteq \Gamma_t$, so by Lemma 18 we have $\Gamma^i \wedge \Gamma^j \to \Delta'' \;\hat{\in}\; N_t$. Since R_t^* is a congruence, we have $R_t^* \not\models l^i[l^j]_p \not\approx r^i$, and therefore $R_t^* \not\models \Delta''$ holds. Finally, $>_t$ is a simplification order, so $l^i \not\approx r^i >_t l^i[l^j]_p$; together with $l^i \approx r^i >_t \Delta^i$ and $l^i \approx r^i >_t \Delta^j$, we have $l^i \approx r^i >_t \Delta''$. But then, Lemma 17 implies $R_t^* \models \Delta''$, which is a contradiction. □

Lemma 22. *For each clause $\Gamma \to \Delta$ with $\Gamma \to \Delta \,\hat{\in}\, N_t$, we have $R_t^* \models \Delta$.*

Proof. Apply Lemma 17 for $i = n + 1$ and Lemma 21. □

Lemma 23. *For each generative clause $\Gamma^i \to \Delta^i \vee l^i \approx r^i$, disjunction Δ^i does not contain a literal of the form $s \not\approx s$.*

Proof. For the sake of a contradiction, let us assume that clause $C^i = \Gamma^i \to \Delta^i \vee l^i \approx r^i \in N_t$ is generative and that $s \not\approx s \in \Delta^i$ holds for some term s. By Lemma 19, we have that $\Gamma^i \to \big(\Delta^i \setminus \{s \not\approx s\}\big) \vee l^i \approx r^i \,\hat{\in}\, N_t$. If this clause satisfies condition 1 of Definition 4, then C^i cannot be generative due to condition R1; hence, this clause satisfies condition 2 of Definition 4 and so $\Gamma \to \Delta \in N_t$ holds for some $\Gamma \subseteq \Gamma^i$ and some $\Delta \subseteq \big(\Delta^i \setminus \{s \not\approx s\}\big) \cup \{l^i \approx r^i\}$. Now Lemma 16 implies $R_t^* \not\models \Delta^i$; moreover, by condition R1, we have $(R_t^{i-1})^* \not\models \Delta^i \vee l^i \approx r^i$. We have the following two possibilities.

- $l^i \approx r^i \in \Delta$. Let j be the index of clause $\Gamma \to \Delta$ in the sequence of clauses from Appendix C.2.2; clearly, $j < i$, and so $(R_t^{j-1})^* \subseteq (R_t^{i-1})^*$. Consider an arbitrary literal $L \in \Delta$: if L is of the form $l \approx r$, then $(R_t^{i-1})^* \not\models L$ clearly implies $(R_t^{j-1})^* \not\models L$; and if L is of the form $l \not\approx r$, then Lemma 15 applied for i ensures $R_t^* \not\models l \not\approx r$, and so Lemma 15 applied again for j ensures $(R_t^{j-1})^* \not\models l \not\approx r$. Thus, we have $(R_t^{j-1})^* \not\models \Delta$, and so $\Delta \subseteq \Delta^i$ clearly ensures that $\Gamma \to \Delta$ generates $l^i \Rightarrow r^i$. This, however, contradicts our assumption that C^i is generative.
- $l^i \approx r^i \notin \Delta$. But then, we have $\Delta \subseteq \Delta^i$, and so $R_t^* \not\models \Delta$. However, by Lemma 21 we have $R_t^* \models \Delta$, which is a contradiction. □

Lemma 24. $R_t^* \not\models \Gamma_t \to \Delta_t$.

Proof. For $R_t^* \models \Gamma_t$, note that condition L3 ensures $\Gamma_t \to A \,\hat{\in}\, N_t$, and so Lemma 22 ensures $R_t^* \models A$ for each atom $A \in \Gamma_t$.

For $R_t^* \not\models \Delta_t$, assume for the sake of a contradiction that a literal $L \in \Delta_t$ exists such that $R_t^* \models L$. Then, a generative clause $C^i = \Gamma^i \to \Delta^i \vee l^i \approx r^i \in N_t$ and a position p exist such that $L|_p = l^i$; let $\Delta = \Delta^i \vee l^i \approx r^i$. Since $>_t$ is a simplification order and $l^i >_t r^i$, we have $L \geq_t l^i \approx r^i$; but then, since $l^i \approx r^i >_t \Delta^i$, we have $L \geq_t \Delta$. We next consider an arbitrary literal $l \bowtie r \in \Delta$ with $\bowtie \in \{\approx, \not\approx\}$ and $l \geq_t r$; by the observations made thus far, $L \geq_t l \bowtie r$ holds. By condition O2, one of the following holds.

1. $l \bowtie r \in \{t \approx t, t' \approx t'\}$. But then C^i is not generative by condition R1.
2. $l \bowtie r \in \{t \not\approx t, t' \not\approx t'\}$. But then C^i is not generative by Lemma 23.
3. $l \bowtie r = t \approx t'$. But then $t \approx t' \in \Delta_t$ by condition L2.
4. $l \bowtie r = t \not\approx t'$. By Lemma 20, both t and t' are irreducible by R_t, so $R_t^* \models t \not\approx t'$; by Lemma 15, then $(R_t^{i-1})^* \models t \not\approx t'$; but then C^i is not generative by condition R1.
5. $l \approx r \in \Delta_t$ where $r = \mathsf{true}$.

Thus, 3 and 5 are the only possible cases, and they both satisfy $l \bowtie r \in \Delta_t$. Since $l \bowtie r$ was arbitrarily chosen from Δ_t, we have $\Delta \subseteq \Delta_t$. But then, $\Gamma^i \subseteq \Gamma_t$ implies that $\Gamma_t \to \Delta_t \,\hat{\in}\, N_t$ holds, which contradicts condition L1. □

C.3 Unfolding the Context Structure

We now present the construction of the rewrite system R, which is obtained by unfolding the context structure $\mathcal{D}$ and applying the model fragment construction from Appendix C.2 at each step. We use structural induction over the a-terms of HU. We define several partial functions: function X maps a term t to a context $X_t \in \mathcal{V}$; functions Γ and Δ assign to a term t a conjunction of atoms Γ_t and a disjunction of literals Δ_t, respectively; and function R maps each term into a model fragment R_t for t, X_t, Γ_t, and Δ_t.

M1. For the base case, we consider the constant c.

$$X_c = q \tag{115}$$
$$\Gamma_c = \Gamma_Q \sigma_c \tag{116}$$
$$\Delta_c = \Delta_Q \sigma_c \tag{117}$$
$$R_c = \text{the model fragment for } c,\ q,\ \Gamma_c, \text{ and } \Delta_c \tag{118}$$

M2. For the inductive step, assume that $X_{t'}$ has already been defined, and consider an arbitrary function symbol $f \in \Sigma_{\mathsf{a}}^F$ such that $f(t')$ is irreducible by $R_{t'}$. Let $u = X_{t'}$ and $t = f(t')$. We have two possibilities.

M2.a. Term t occurs in $R_{t'}$. Then, term $t = f(t')$ was generated in $R_{t'}$ by some ground clause $C = \Gamma \to \Delta \vee L \in N_{t'}$ such that $L >_t \Delta$ and $f(t')$ occurs in L. By the definition of N_t, then a clause $C' = \Gamma' \to \Delta' \vee L' \in \mathcal{S}_u$ exists such that $C = C'\sigma_{t'}$ and L' contains $f(x)$; moreover, $L >_{t'} \Delta$ implies $\Delta' \not\geq_u L'$. The Succ and Core rules are not applicable to $\mathcal{D}$, so we can choose a context $v \in \mathcal{V}$ such that $\langle u, v, f\rangle \in \mathcal{E}$ and $A \to A \,\hat{\in}\, \mathcal{S}_v$ for each $A \in K_2$, where K_2 is as in the Succ rule. We define the following:

$$X_t = v \tag{119}$$
$$\Gamma_t = R^*_{t'} \cap \mathsf{Su}_t \tag{120}$$
$$\Delta_t = \mathsf{Pr}_t \setminus R^*_{t'} \tag{121}$$
$$R_t = \text{the model fragment for } t,\ v,\ \Gamma_t, \text{ and } \Delta_t \tag{122}$$

M2.b. Term t does not occur in $R_{t'}$. Then, let $R_t = \{t \Rightarrow c\}$, and we do not define any other functions for t.

Finally, let R be the rewrite system defined by $R = \bigcup_t R_t$ where the union ranges over all R_t that have been defined above.

Lemma 25. *The model fragments R_c and R_t constructed in lines* (118) *and* (122) *satisfy conditions L1 through L3 in Appendix C.2.*

Proof. The proof is by induction on the structure of terms $t \in \mathsf{dom}(X)$. For $t = c$, if condition L1 were not satisfied, then $v = q$ due to (115), (116), and (117) all imply $\Gamma_Q \to \Delta_Q \,\hat{\notin}\, \mathcal{S}_q$, which contradicts our assumption in Theorem 9. In addition, condition L2 follows trivially by (117), and condition L3 holds because of (116) and condition C3 of Theorem 9. We next assume that the lemma holds for some term $t' \in \mathsf{dom}(X)$, and we consider an arbitrary term

t of the form $t = f(t')$; let $u = X_{t'}$ and $v = X_t$. Note that terms t and t' are irreducible by $R_{t'}$ due to condition M2. Moreover, Su_t contains atoms of the form $B(t)$, $S(t, t')$, and $S(t', t)$, and so each atom in Su_t is irreducible by $R_{t'}$. But then, by (120), Γ_t contains atoms of $R^*_{t'}$ that are irreducible by $R_{t'}$; for convenience, assume that $\Gamma_t = \{A_1, \ldots, A_n\}$, where subscripts do not necessarily indicate the position of the clause in sequence of clauses from Appendix C.2.2. Then, each atom $A_i \in \Gamma_t$ is generated by a clause satisfying (123). By the definition of $N_{t'}$, then there exists a clause satisfying (124).

$$\Gamma_i \to \Delta_i \vee A_i \in N_{t'} \quad \text{with} \quad A_i >_t \Delta_i \tag{123}$$

$$\Gamma'_i \to \Delta'_i \vee A'_i \in \mathcal{S}_u \quad \Gamma_i = \Gamma'_i\sigma_{t'}, \quad \Delta_i = \Delta'_i\sigma_{t'}, \quad A_i = A'_i\sigma_{t'}, \quad \text{and} \quad \Delta'_i \not\succeq_u A'_i \tag{124}$$

To prove condition L1, assume for the sake of a contradiction that $\Gamma_t \to \Delta_t \mathrel{\hat{\in}} N_t$ holds. Since $\Delta_t \subseteq \mathsf{Pr}_t$ holds due to (121), only condition condition 2 of Definition 4 can hold—that is, set N_t then contains a clause

$$\bigwedge_{i=1}^{m} A_i \to \bigvee_{i=m+1}^{m+n} L_i \quad \begin{array}{l} \text{with } \{\, A_i \mid 1 \leq i \leq m \,\} \subseteq \Gamma_t \subseteq R^*_{t'} \cap \mathsf{Su}_t \\ \text{and } \{\, L_i \mid m+1 \leq i \leq m+n \,\} \subseteq \Delta_t \subseteq \mathsf{Pr}_t; \end{array} \tag{125}$$

to simplify indexing, we assume w.l.o.g. that $A_1, \ldots, A_m$ are the first m atoms from Γ_t. By the definition of N_t, set $\mathcal{S}_v$ contains a clause

$$\bigwedge_{i=1}^{m} A'_i \to \bigvee_{i=m+1}^{m+n} L'_i \quad \begin{array}{l} \text{with } A_i = A'_i\sigma_t \text{ for } 1 \leq i \leq m \\ \text{and } \; L_i = L'_i\sigma_t \text{ and } L'_i \in \mathsf{Pr}(\mathcal{O}) \text{ for } m+1 \leq i \leq m+n. \end{array} \tag{126}$$

Now each A_i with $1 \leq i \leq m$ is generated by a ground clause (123), and the latter is obtained from the corresponding nonground clause (124). The Pred rule is not applicable to (124) and (126) so (127) holds; together with Lemma 18, this ensures (128).

$$\bigwedge_{i=1}^{m} \Gamma'_i \to \bigvee_{i=1}^{m} \Delta'_i \vee \bigvee_{i=m+1}^{m+n} L'_i\sigma \mathrel{\hat{\in}} \mathcal{S}_u \quad \text{for } \sigma = \{x \mapsto f(x),\ y \mapsto x\} \tag{127}$$

$$\bigwedge_{i=1}^{m} \Gamma_i \to \bigvee_{i=1}^{m} \Delta_i \vee \bigvee_{i=m+1}^{m+n} L_i \mathrel{\hat{\in}} N_{t'} \tag{128}$$

By Lemma 16, we have $R^*_{t'} \not\models \Delta_i$ for each $1 \leq i \leq m$; moreover, (121) ensures $R^*_{t'} \not\models \Delta_t$, which implies $R^*_{t'} \not\models L_i$ for each $m+1 \leq i \leq m+n$; however, this contradicts (128) and Lemma 22.

Next we show that condition L2 holds. Since $t \neq c$, we have $\Delta_t = \mathsf{Pr}_t \setminus R^*_{t'}$ by (121), and hence $\Delta_t \subseteq \mathsf{Pr}_t$ holds. Furthermore, $\{t \approx t'\} \subseteq \mathsf{Pr}_t$ holds by (107); moreover, t is irreducible by $R_{t'}$ by condition M2, and t' is irreducible by $R_{t'}$ since Lemma 20 holds for $R_{t'}$ by the induction hypothesis. Hence, $R^*_{t'} \not\models t \approx t'$ holds, so we have $\{t \approx t'\} \subseteq \Delta_t$, as required.

Finally, we show that condition L3 holds. Consider an arbitrary atom $A_i \in \Gamma_t$, let (123) be the clause that generates A_i in $R_{t'}$, and let (124) be the corresponding nonground clause. Since $A_i \in \mathsf{Su}_t$, atom A'_i is of the form $A''_i\sigma$, where σ is the substitution from the Succ rule; but then, $A''_i \in K_2$, where K_2 is as specified in the Succ rule. In condition M2.a we chose v so that the Succ rule is satisfied, and therefore $A''_i \to A''_i \mathrel{\hat{\in}} \mathcal{S}_v$; but then, since $A''_i\sigma_t = A_i$, we have $A_i \to A_i \mathrel{\hat{\in}} N_t$, as required for condition L3. □

Lemma 26. *The rewrite system R is Church-Rosser.*

Proof. We show that R is terminating and left-reduced, and thus Church-Rosser. In the proof of the former, we use a total simplification order $\rhd$ on all ground a- and p-terms defined as follows. We extend the total order $\succcurlyeq$ on function symbols to all a- and p-symbols in an arbitrary way, but ensuring that constant true is smallest in the order; then, let $\rhd$ be a *lexicographic path order* (Baader & Nipkow, 1998) over such $\succcurlyeq$. It is well known that such $\rhd$ is a simplification order, and that it satisfies the following properties for each a-term t with predecessor t' (if it exists), all function symbols $f, g \in \Sigma_{\mathsf{a}}^F$, and each p-term A:

- $f(t) \rhd t \rhd t'$,
- $f \succcurlyeq g$ implies $f(t) \rhd g(t)$, and
- $A \rhd \mathsf{true}$.

Thus, conditions 1 and 2 of Definition 3 and the manner in which context orders are grounded in Appendix C.2.1 clearly ensure that, for each a-term $t \in \mathsf{dom}(X)$ and for all terms s_1 and s_2 from the a-neighbourhood of t with $s_1 >_t s_2$, we have $s_1 \rhd s_2$.

We next show that R is terminating by arguing that each rule in R is embedded in $\rhd$. To this end, consider an arbitrary rule $l \Rightarrow r \in R$. Clearly, a term $t \in \mathsf{dom}(R)$ exists such that $l \Rightarrow r \in R_t$. This rule is obtained from a head literal $l \approx r$ of a clause in N_t, and condition R2 of the definition of R_t ensures that $l >_t r$. Moreover, $l \approx r$ is obtained by grounding a context literal with σ_t, so we have the following possible forms of $l \approx r$.

- Terms l and r are both from the a-neighbourhood of t. Then, $l >_t r$ implies $l \rhd r$.
- We have $l \approx r = A \approx \mathsf{true}$ for A a p-term. Then, $A \rhd \mathsf{true}$ since true is smallest in $\succcurlyeq$.

We next show that R is left-reduced. For the sake of a contradiction, assume that there exist a term s and a rule $l \Rightarrow r \in R_s \subseteq R$ such that l is reducible by $R' = R \setminus \{l \Rightarrow r\}$. This rule is generated by an equality in the head of a generative clause, and so it is of the form $A \Rightarrow \mathsf{true}$ where A contains s and/or s', or of the form $f(s) \Rightarrow g(s)$, $f(s) \Rightarrow s$, $f(s) \Rightarrow s'$, or $s \Rightarrow s'$, for s' the predecessor of s (if one exists). Let p be the 'deepest' position at which some rule in R' reduces l (i.e., no rule in R' reduces l at position below p), and let $l' \Rightarrow r' \in R'$ be the rule that reduces l at position p; thus, $l' = l|_p$. If l' is a p-term, then $l = l' = A$ and $r = r' = \mathsf{true}$, but then $l \Rightarrow r \in R'$, which contradicts our definition of R'. Hence, l' is an a-term and, due to the possible forms of $l \Rightarrow r$, it is a (not necessarily proper) subterm of s. By the definition of R, there exists a term t such that $l' \Rightarrow r' \in R_t$. But then, condition M2 ensures that $R_{l'}$ is not defined since l' is reducible by R_t. Consequently, rewrite system R_s is not defined either since it contains l'. □

Lemma 27. *For each term t, each $f \in \Sigma_{\mathsf{a}}^F$, and each atom $A \in \mathsf{Su}_t \cup \mathsf{Pr}_{f(t)} \cup \mathsf{Ref}_t$ such that $A \in R^*$ and all a-terms in A are irreducible by R, we have $R_t^* \models A$.*

Proof. Let t be a term, let $f \in \Sigma_{\mathsf{a}}^F$ be a function symbol, and let $A \in \mathsf{Su}_t \cup \mathsf{Pr}_{f(t)} \cup \mathsf{Ref}_t$ be an atom such that all a-terms in A are irreducible by R; the latter and $A \in R^*$ ensure that $A \Rightarrow \mathsf{true} \in R$. We next consider the possible forms of A.

Assume $A \in \mathsf{Su}_t$. By the definition of Su_t in (106) and the fact that $\mathsf{Su}(\mathcal{O})$ contains only atoms of the form $B(x)$, $S(x,y)$, and $S(y,x)$, atom A can be of the form $B(t)$, $S(t,t')$, or $S(t',t)$, for t' the predecessor of t (if it exists). Due to the form of context literals in Definition 1 and the definition of grounding from Appendix C.2, atom A can occur only in N_t or $N_{t'}$; therefore, we have $R^*_t \models A$ or $R^*_{t'} \models A$. Now assume $R^*_{t'} \models A$. Due to $A \in \mathsf{Su}_t$ and the definition of Γ_t in (120), we have $A \in \Gamma_t$. Lemma 24 ensures that $R^*_t \not\models \Gamma_t \to \Delta_t$. But then, we have $R^*_t \models A$, as required.

Assume $A \in \mathsf{Pr}_{f(t)}$. By the definition of $\mathsf{Pr}_{f(t)}$ in (107) and the fact that $\mathsf{Pr}(\mathcal{O})$ contains only atoms of the form $B(y)$, $S(y,x)$, and $S(x,y)$, atom A can be of the form, $B(t)$, $S(t,f(t))$, or $S(f(t),t)$; note that $x \approx y \in \mathsf{Pr}(\mathcal{O})$ is not an atom and is therefore not relevant to this analysis. Due to the form of context literals in Definition 1 and the definition of grounding from Appendix C.2, atom A can occur only in N_t or $N_{f(t)}$; therefore, $R^*_t \models A$ or $R^*_{f(t)} \models A$. Assume for the sake of a contradiction that $R^*_t \not\models A$, but $R^*_{f(t)} \models A$. Due to $A \in \mathsf{Pr}_{f(t)}$ and the definition of $\Delta_{f(t)}$ in (121), we have $A \in \Delta_{f(t)}$; due to Lemma 24, we have $R^*_{f(t)} \not\models \Gamma_{f(t)} \to \Delta_{f(t)}$; therefore, we have $R^*_{f(t)} \not\models A$, which is a contradiction.

Assume $A \in \mathsf{Ref}_t$. Due to the form of context literals in Definition 1 and the definition of grounding from Appendix C.2, atom A can occur only in N_t, and so $R^*_t \models A$. □

Lemma 28. *Let s_1 and s_2 be both DL-a-terms or both DL-p-terms, and let τ be a ground substitution irreducible by R such that x is in the domain of τ, terms $s_1\tau$ and $s_2\tau$ are both ground, and, for each z_i in the domain of τ, term $\tau(z_i)$ is in the a-neighbourhood of $\tau(x)$. Then, for $\bowtie \in \{\approx, \not\approx\}$, if $R^*_{\tau(x)} \models s_1\tau \bowtie s_2\tau$, then $R^* \models s_1\tau \bowtie s_2\tau$.*

Proof. Let s_1 and s_2 and τ be as stated above, let $t = \tau(x)$, and let t' be the predecessor of t (if it exists). Since t is irreducible by R, rewrite system R_t has been defined in Appendix C.3. We next consider the possible forms of $\bowtie$.

- Assume $\bowtie = \approx$. But then, $R_t \subseteq R$ and $R^*_t \models s_1\tau \approx s_2\tau$ imply $R^* \models s_1\tau \approx s_2\tau$.
- Assume $\bowtie = \not\approx$. Let s'_1 and s'_2 be the normal forms of $s_1\tau$ and $s_2\tau$, respectively, w.r.t. R_t. Due to the shape of DL-literals, s_1 and s_2 can be of the form $f(x)$, x, or z_i; therefore, terms $s_1\tau$ and $s_2\tau$ are of the form $f(t)$, t, or t', and terms s'_1 and s'_2 are of the form $g(t)$, t, or t'. Term t is irreducible by R, and thus t' is irreducible by R as well. Furthermore, $g(t)$ is irreducible by R_t and R_t is the only rewrite system where $g(t)$ can occur on the left-hand side of a rewrite rule, so therefore $g(t)$ is irreducible by R as well. But then, s'_1 and s'_2 are the normal forms of $s_1\tau$ and $s_2\tau$, respectively, w.r.t. R; thus, $R^* \models s'_1 \not\approx s'_2$, and therefore $R^* \models s_1\tau \not\approx s_2\tau$ holds, as required. □

C.4 The Completeness Claim

Lemma 29. *For each DL-clause $\Gamma \to \Delta \in \mathcal{O}$, we have $R^* \models \Gamma \to \Delta$.*

Proof. Consider an arbitrary DL-clause $\Gamma \to \Delta \in \mathcal{O}$ of the following form:

$$\textstyle\bigwedge_{i=1}^{n} A_i \to \Delta \tag{129}$$

Let τ' be an arbitrary substitution such that $\Gamma\tau' \to \Delta\tau'$ is ground, and let τ be the substitution obtained from τ' by replacing each ground term with its normal form w.r.t. R; by

Lemma 26, such τ is unique. Since R^* is a congruence, we have $R^* \models \Gamma\tau' \to \Delta\tau'$ if and only if $R^* \models \Gamma\tau \to \Delta\tau$. We next assume that $R^* \models \Gamma\tau$, and show that $R^* \models \Delta\tau$ holds as well.

Consider an arbitrary atom $A_i \in \Gamma$. By the definition of DL-clauses, A_i is of the form $B(x)$, $S(x,x)$, $S(x,z_j)$, or $S(z_j,x)$. Substitution τ is irreducible by R, and so all a-terms in $A_i\tau$ are irreducible by R; but then, $A_i\tau \in R^*$ clearly implies $A_i\tau \Rightarrow \mathsf{true} \in R$. Each such rule is obtained from a generative clause so $A_i\tau$ is of the form $B(t)$, $S(t,t)$, $S(t,f(t))$, $S(f(t),t)$, $S(t,t')$, or $S(t',t)$, where $t = \tau(x)$ and t' is the predecessor of t (if it exists). We next prove that $A_i\tau \in \mathsf{Su}_t \cup \mathsf{Pr}_{f(t)} \cup \mathsf{Ref}_t$ holds by considering the possible forms of A_i.

- $A_i = B(x)$, so $A_i\tau = B(t)$. Then, $B(x) \in \mathsf{Su}(\mathcal{O})$, and so $B(t) \in \mathsf{Su}_t$ holds.
- $A_i = S(x,x)$, so $A_i\tau = S(t,t)$. Then, we clearly have $S(t,t) \in \mathsf{Ref}_t$.
- $A_i = S(x,z_j)$, so $A_i\tau$ is of the form $S(t,t')$ or $S(t,f(t))$. Then, $S(x,y) \in \mathsf{Su}(\mathcal{O})$, and so $S(t,t') \in \mathsf{Su}_t$ holds; moreover, $S(y,x) \in \mathsf{Pr}(\mathcal{O})$, and so $S(t,f(t)) \in \mathsf{Pr}_{f(t)}$ holds.
- $A_i = S(z_j,x)$, so $A_i\tau$ is of the form $S(t',t)$ or $S(f(t),t)$. Then, $S(y,x) \in \mathsf{Su}(\mathcal{O})$, and so $S(t',t) \in \mathsf{Su}_t$ holds; moreover, $S(x,y) \in \mathsf{Pr}(\mathcal{O})$, and so $S(f(t),t) \in \mathsf{Pr}_{f(t)}$ holds.

Lemma 27 then implies $A_i\tau \in R_t$, and so N_t contains a generative clause of the form (130). Now let $v = X_t$; by the definition of N_t, set $\mathcal{S}_v$ contains a clause of the form (131).

$$\Gamma_i \to \Delta_i \vee A_i \text{ with } A_i >_t \Delta_i \text{ and } \Gamma_i \subseteq \Gamma_t \tag{130}$$

$$\Gamma'_i \to \Delta'_i \vee A'_i \text{ with } \Delta'_i \not\geq_v A'_i \text{ and } \Gamma'_i\sigma_t = \Gamma_i,\ \Delta'_i\sigma_t = \Delta_i, \text{ and } A'_i\sigma_t = A_i \tag{131}$$

The Hyper rule is not applicable to (129) and (131), and therefore (132) holds, where σ is the substitution obtained from τ by replacing each occurrence of t (possibly nested in another term) with x. Finally, Lemma 18 ensures that (133) holds as well.

$$\bigwedge_{i=1}^{n} \Gamma'_i \to \Delta\sigma \vee \bigvee_{i=1}^{n} \Delta'_i \mathrel{\hat{\in}} \mathcal{S}_v \tag{132}$$

$$\bigwedge_{i=1}^{n} \Gamma_i \to \Delta\tau \vee \bigvee_{i=1}^{n} \Delta_i \mathrel{\hat{\in}} N_t \tag{133}$$

Now (133) and Lemma 22 imply $R^*_t \models \Delta\tau \vee \bigvee_{i=1}^{n} \Delta_i$, but Lemma 16 implies $R^*_t \not\models \Delta_i$; therefore, we have $R^*_t \models \Delta\tau$. Finally, Lemma 28 ensures $R^* \models \Delta\tau$, as required. □

Lemma 30. $R^* \not\models \Gamma_Q \to \Delta_Q$.

Proof. The claim is equivalent to proving $R^* \not\models \Gamma_c \to \Delta_c$. Lemma 24 implies $R^*_c \not\models \Gamma_c \to \Delta_c$, and so $R^*_c \models \Gamma_c$ and $R^*_c \not\models \Delta_c$ hold. The former observation and Lemma 28 ensure that $R^* \models \Gamma_c$ holds. Furthermore, for each atom $B(x) \in \Delta_Q$, Definition 2 ensures $B(y) \in \mathsf{Pr}(\mathcal{O})$, and so $B(c) \in \mathsf{Pr}_{f(c)}$ holds for each $f \in \Sigma^F_{\mathsf{a}}$; hence, the contrapositive of Lemma 27 ensures $R^* \not\models B(c)$. Finally, for each atom $S(x,x) \in \Delta_Q$, we have $S(c,c) \in \mathsf{Ref}_c$; hence, the contrapositive of Lemma 27 ensures $R^* \not\models S(c,c)$. Consequently, $R^* \not\models \Delta_c$ holds, as required. □

Appendix D. Proof of Proposition 10

Proposition 10. *Algorithm 1 terminates whenever the expansion strategy introduces finitely many contexts in the algorithm's run. The algorithm runs in worst-case exponential time in the size of $\mathcal{O}$ if the number of introduced contexts is at most exponential in the size of $\mathcal{O}$.*

Proof. Let k be the number of DL-clauses on $\mathcal{O}$, and let m the be the larger of the maximum number of body atoms of a DL-clause in $\mathcal{O}$ and the maximum size of the body of a context clause; both k and m are linear in $\mathcal{O}$ since the number of variables in a context clause is fixed. The number $\wp$ of context clauses that can be constructed using the symbols in $\mathcal{O}$ is at most exponential in the size of $\mathcal{O}$.

For the first claim, if the number of contexts is finite, the total number of context clauses is finite as well. Moreover, once an inference is applied, its preconditions never become satisfied again. Hence, the number of possible inferences is finite and each inference is performed just once, so Algorithm 1 terminates.

For the second claim, assume that the strategy can introduce at most n contexts, where n is exponential in the size of $\mathcal{O}$. The number of possible inferences is bounded as follows.

- The number of distinct inferences by the Hyper rule within each context is bounded by $k \cdot \wp^m$. Hence, the total number of inferences is bounded by $k \cdot \wp^m \cdot n$, which is exponential in the size of $\mathcal{O}$.
- The number of clauses participating in each distinct inference by a rule other than Pred or Succ is constant, so an exponential bound on the number of inferences by these rules can be obtained analogously to the Hyper rule.
- The Pred rule can be applied to any pair of contexts, and each inference involves one clause from one context and at most m clauses from the other context. Hence, the number of distinct inferences is bounded by $\wp \cdot \wp^m \cdot n^2$, which is exponential in the size of $\mathcal{O}$.
- The Succ rule can be applied to any context. Now consider an application of the rule to a context u, and let clauses $\Gamma \to \Delta \vee A$ and $\Gamma' \to \Delta' \vee A'\sigma$ play the roles as specified in Table 2. The preconditions of the Succ rule can become satisfied either when a clause $\Gamma \to \Delta \vee A$ is added to $\mathcal{S}_u$, or when a clause $\Gamma' \to \Delta' \vee A'\sigma$ is added to $\mathcal{S}_u$ and thus changes the set K_2. Hence, the rule can become applicable at most $\wp^2 \cdot n$ times, which is exponential in the size of $\mathcal{O}$. □

Appendix E. Proof of Proposition 11

Proposition 11. *On $\mathcal{ELH}$ ontologies and queries of the form $B_1(x) \to B_2(x)$, Algorithm 1 runs in polynomial time in the size of $\mathcal{O}$ with either the cautious or the eager strategy.*

Proof. Consider an $\mathcal{ELH}$ ontology represented as a set $\mathcal{O}$ of DL-clauses of type DL1 with $n \leq m \leq n+1$, DL2 with $n = 1$, DL3, and DL7 from Section 2. In addition, assume that a query is of the form $B_1(x) \to B_2(x)$, so Algorithm 1 initialises the core of q to $B_1(x)$.

We first consider applying Algorithm 1 to $\mathcal{O}$ with the cautious strategy. By a straightforward induction on the application of the rules from Table 2, one can see that each derived

context clause is of the form (134)–(139).

$$\top \to \bot \tag{134}$$
$$\top \to B(x) \tag{135}$$
$$\top \to S(x, f(x)) \tag{136}$$
$$\top \to B(f(x)) \tag{137}$$
$$S(y, x) \to B(y) \tag{138}$$
$$S(y, x) \to S'(y, x) \tag{139}$$

Moreover, the cautions strategy introduces at most one context with the core of the form $B(x)$ for each atomic concept B. Since the number of atomic concepts and roles is linear in the size of $\mathcal{O}$ and the inference rules can be applied in polynomial time, the algorithm runs in polynomial time.

We next consider applying Algorithm 1 to $\mathcal{O}$ with the eager strategy. Again, by a straightforward induction on the application of the rules from Table 2, one can see that each derived context clause is of the form (134)–(141).

$$\top \to S(y, x) \tag{140}$$
$$\top \to B(y) \tag{141}$$

Moreover, the cautious strategy introduces at most one context with the core of the form $B(x)$, $S(y, x)$, or $B(x) \wedge S(y, x)$. Again, the algorithm clearly runs in polynomial time. □

Appendix F. Proof of Proposition 12

Proposition 12. *If the ontology is in $\mathcal{EL}$, the context structure is initialised by a context v_B with $\mathsf{core}_{v_B} = \{B(x)\}$ for each atomic concept B, the* Hyper *rule is applied eagerly, and the cautious strategy is used, then numbers of inferences in step A3 of Algorithm 1 and of the calculus by Baader et al. (2005) are in a linear relationship.*

Proof. Consider an $\mathcal{EL}$ ontology represented as a set $\mathcal{O}$ of DL-clauses of type DL1 with $n \leq m \leq n + 1$, DL2 with $n = 1$ and DL3. Note that each function symbol f occurs in exactly one pair of DL-clauses of type DL2; we say that f *belongs* to the atomic role S from the corresponding DL-clause of type DL2. For simplicity, we slightly abuse the notation and use $\mathcal{O}$ to also denote the ontology written in the description logic syntax.

The algorithm by Baader et al. (2005) computes a mapping S that associates each atomic concept B with a set $\mathsf{S}(B)$ of consisting of atomic concepts and $\bot$, and a mapping R that associates each role S with a set $\mathsf{R}(S)$ of pairs of atomic concepts. Mapping S is initialised by setting $\mathsf{S}(B) = \{B\}$, and then the inference rules summarised in Table 3 are applied (since the ontology is in $\mathcal{EL}$, rules CR6–CR11 are never applicable). This procedure ensures that $\mathcal{O} \models B \sqsubseteq B'$ holds for all B and $B' \in \mathsf{S}(B)$.

We now show that each application of rules CR1–CR5 corresponds to at most three applications of the inference rules from Table 2. To this end, we show that there is the following correspondence between mappings S and R and the context structure for all atomic concepts B_1 and B_2 and each atomic role S:

Table 3: Rules of the $\mathcal{EL}$ Calculus by Baader et al. (2005)

CR1	If $B_1 \in \mathsf{S}(B)$, $B_1 \sqsubseteq B_2 \in \mathcal{O}$, and $B_2 \notin \mathsf{S}(B)$, then add B_2 to $\mathsf{S}(B)$.
CR2	If $\{B_1, B_2\} \subseteq \mathsf{S}(B)$, $B_1 \sqcap B_2 \sqsubseteq B_3 \in \mathcal{O}$, and $B_3 \notin \mathsf{S}(B)$, then add B_3 to $\mathsf{S}(B)$.
CR3	If $B_1 \in \mathsf{S}(B)$, $B_1 \sqsubseteq \exists S.B_2 \in \mathcal{O}$, and $\langle B, B_2 \rangle \notin \mathsf{R}(S)$, then add $\langle B, B_2 \rangle$ to $\mathsf{R}(S)$.
CR4	If $\langle B_1, B_2 \rangle \in \mathsf{R}(S)$, $B_3 \in \mathsf{S}(B_2)$, $\exists S.B_3 \sqsubseteq B_4 \in \mathcal{O}$, and $B_4 \notin \mathsf{S}(B_1)$ then add B_4 to $\mathsf{S}(B_1)$.
CR5	If $\langle B_1, B_2 \rangle \in \mathsf{R}(S)$, $\bot \in \mathsf{S}(B_2)$, and $\bot \notin \mathsf{S}(B_1)$ then add $\bot$ to $\mathsf{S}(B_1)$.

- $B_2 \in \mathsf{S}(B_1)$ implies $\top \to B_2(x) \in \mathcal{S}_{v_{B_1}}$,
- $\bot \in \mathsf{S}(B_1)$ implies $\top \to \bot \in \mathcal{S}_{v_{B_1}}$, and
- $\langle B_1, B_2 \rangle \in \mathsf{R}(S)$ implies $\{\top \to S(x, f(x)), \top \to B_2(f(x))\} \subseteq \mathcal{S}_{v_{B_1}}$, $\langle v_{B_1}, v_{B_2}, f \rangle \in \mathcal{E}$, and $S(y, x) \to S(y, x) \in \mathcal{S}v_{B_2}$ for some function symbol f that S belongs to.

We prove the claim by induction on the application of the rules CR1–CR5. The context structure initialisation ensures that these properties hold for the induction base. For the induction step, we consider all possible applications of the rules CR1–CR5 shown in Table 3.

- An application of the rule CR1 or CR2 corresponds straightforwardly to an application of the Hyper rule.
- Consider an application of the rule CR3. Then, the Hyper rule adds $\top \to S(x, f(x))$ and $\top \to B_2(x, f(x))$ for $\mathcal{S}_{v_{B_1}}$ for f the function symbol from the corresponding DL-clauses of type DL2; since the Hyper rule is applied eagerly, both clauses are derived before the Succ rule is applied. Since the cautious strategy is used, the Succ rule next introduces $\langle v_{B_1}, v_{B_2}, f \rangle \in \mathcal{E}$, and it adds $S(y, x) \to S(y, x)$ to $\mathcal{S}_{v_{B_2}}$.
- Consider an application of the rule CR4. Then we have $\top \to S(x, f(x)) \in \mathcal{S}_{v_{B_1}}$, $\langle v_{B_1}, v_{B_2}, f \rangle \in \mathcal{E}$, and $\{S(y, x) \to S(y, x), \top \to B_3(x)\} \subseteq \mathcal{S}_{v_{B_2}}$ for some function symbol f by the induction assumption, so the Hyper rule adds $S(y, x) \to B_4(y)$ to $\mathcal{S}_{v_{B_2}}$, and then the Pred rule adds $\top \to B_4(x)$ to $\mathcal{S}_{v_{B_1}}$.
- Consider an application of the rule CR5. The induction assumption then ensures that $\top \to S(x, f(x)) \in \mathcal{S}_{v_{B_1}}$, $\langle v_{B_1}, v_{B_2}, f \rangle \in \mathcal{E}$, and $\top \to \bot \in \mathcal{S}_{v_{B_2}}$ hold for some function symbol f. Thus, the Pred rule adds $\top \to \bot$ to $\mathcal{S}_{v_{B_1}}$, as required.

Conversely, we show that, in each sequence of inferences of the rules from Table 2, we can group the inferences so that each group contains at most three inferences and corresponds to one application of a rule CR1–CR5. To this end, we also show that there is the following correspondence between the context structure and the mappings S and R for all atomic concepts B_1 and B_2 and each atomic role S:

- all context clauses derived by the rules from Table 2 are of the form (134)–(138), or of the form (139) with $S = S'$,
- $S(y,x) \to S(y,x) \in \mathcal{S}_{v_{B_1}}$ or $S(y,x) \to B_2(y) \in \mathcal{S}_{v_{B_1}}$ implies that there exist a context v_{B_3} and function symbol f such that $\top \to S(x, f(x)) \in \mathcal{S}_{v_{B_3}}$ and $\langle v_{B_3}, v_{B_1}, f\rangle \in \mathcal{E}$,
- $\top \to B_2(x) \in \mathcal{S}_{v_{B_1}}$ implies $B_2 \in \mathsf{S}(B_1)$, and
- $\langle v_{B_1}, v_{B_2}, f\rangle \in \mathcal{E}$ implies $\langle B_1, B_2\rangle \in \mathsf{R}(S)$ for S the role that f belongs to.

The proof is by induction on the application of the rules from Table 2. The context structure initialisation ensures that these properties hold for the induction base. For the induction step, we consider all possible applications of the rules Table 2.

- An application of the Hyper rule to a DL-clause of type DL1 corresponds straightforwardly to an application of the rule CR1 or CR2.
- The Hyper rule can be applied to DL-clauses $B_1(x) \to S(x, f(x))$ or $B_1(x) \to B_2(f(x))$ of type DL2 to a context clause $\top \to B_1(x) \in \mathcal{S}_{v_B}$. The Hyper rule is applied eagerly, so both $\top \to S(x, f(x))$ and $\top \to B_2(f(x))$ are derived in $\mathcal{S}_{v_B}$. Next, since the cautious strategy is used, the Succ rule ensures $\langle v_B, v_{B_2}, f\rangle \in \mathcal{E}$ and $S(y,x) \to S(y,x) \in \mathcal{S}_{v_{B_2}}$. The induction assumption then ensures $B_1 \in \mathsf{S}(B)$, so these inferences correspond to an application of the rule CR3 that adds $\langle B, B_2\rangle$ to $\mathsf{R}(S)$.
- The Hyper rule can be applied to a DL-clause $S(z_1,x) \wedge B_3(x) \to B_4(y)$ of type DL3 to context clauses $\{S(y,x) \to S(y,x), \top \to B_3(x)\} \subseteq \mathcal{S}_{v_{B_2}}$. The induction assumption ensures that there exists a context B_1 such that $\top \to S(x, f(x)) \in \mathcal{S}_{v_{B_1}}$ and $\langle v_{B_1}, v_{B_2}, f\rangle \in \mathcal{E}$, so the Pred rule derives $\top \to B_4(x) \in \mathcal{S}_{v_{B_1}}$. The induction assumption then eusre $\langle B_1, B_2\rangle \in \mathsf{R}(S)$ and $B_3 \in \mathsf{S}(B_2)$, so these inferences correspond to an application of the rule CR4 that adds B_4 to $\mathsf{S}(B_1)$.
- The Pred can be applied to a context clause $\top \to \bot \in \mathcal{S}_{v_{B_2}}$ and $\langle v_{B_1}, v_{B_2}, f\rangle \in \mathcal{E}$. The induction assumption ensures $\langle B_1, B_2\rangle \in \mathsf{R}(S)$ and $\bot \in \mathsf{S}(B_2)$, so this inference corresponds to an application of the rule CR5 that adds $\bot$ to $\mathsf{S}(B_1)$.

One can verify that the Hyper, Succ, and Pred rules can be applied only as specified above, and the Core, Eq, Ineq, and Factor rules are never applicable, which implies our claim. □

References

Armas Romero, A., Cuenca Grau, B., & Horrocks, I. (2012). MORe: Modular Combination of OWL Reasoners for Ontology Classification. In Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J. X., Hendler, J., Schreiber, G., Bernstein, A., & Blomqvist, E. (Eds.), *Proc. of the 11th Int. Semantic Web Conference (ISWC 2012)*, Vol. 7649 of *LNCS*, pp. 1–16, Boston, MA, USA. Springer.

Baader, F., Brandt, S., & Lutz, C. (2005). Pushing the $\mathcal{EL}$ Envelope. In Kaelbling, L. P., & Saffiotti, A. (Eds.), *Proc. of the 19th Int. Joint Conference on Artificial Intelligence (IJCAI 2005)*, pp. 364–369, Edinburgh, UK. Morgan Kaufmann Publishers.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. F. (Eds.). (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Baader, F., & Nipkow, T. (1998). *Term Rewriting and All That*. Cambridge University Press.

Baader, F., & Sattler, U. (2001). An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, *69*, 5–40.

Baader, F., Lutz, C., & Suntisrivaraporn, B. (2006). Efficient Reasoning in $\mathcal{EL}^+$. In Parsia, B., Sattler, U., & Toman, D. (Eds.), *Proc. of the 19th Int. Workshop on Description Logics (DL 2006)*, Vol. 189 of *CEUR Workshop Proceedings*, Windermere, UK.

Bachmair, L., & Ganzinger, H. (1998). Equational Reasoning in Saturation-Based Theorem Proving. In Bibel, W., & Schmidt, P. (Eds.), *Automated Deduction—A Basis for Applications*, Vol. I, pp. 353–397. Kluwer.

Bachmair, L., & Ganzinger, H. (2001). Resolution Theorem Proving. In Robinson, A., & Voronkov, A. (Eds.), *Handbook of Automated Reasoning*, Vol. I, chap. 2, pp. 19–99. Elsevier Science.

Bate, A., Motik, B., Cuenca Grau, B., Simančík, F., & Horrocks, I. (2016). Extending Consequence-Based Reasoning to $\mathcal{SRIQ}$. In Baral, C., Delgrande, J. P., & Wolter, F. (Eds.), *Proc. of the 15th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2016)*, pp. 187–196, Cape Town, South Africa.

Baumgartner, P., Furbach, U., & Niemelä, I. (1996). Hyper Tableaux. In *Proc. of the European Workshop on Logics in Artificial Intelligence (JELIA '96)*, No. 1126 in LNAI, pp. 1–17, Évora, Portugal. Springer.

de Nivelle, H., Schmidt, R. A., & Hustadt, U. (2000). Resolution-Based Methods for Modal Logics. *Logic Journal of the IGPL*, *8*(3), 265–292.

Demri, S., & de Nivelle, H. (2005). Deciding Regular Grammar Logics with Converse Through First-Order Logic. *Journal of Logic, Language and Information*, *14*(3), 289–329.

Ganzinger, H., & de Nivelle, H. (1999). A Superposition Decision Procedure for the Guarded Fragment with Equality. In *Proc. of the 14th IEEE Symposium on Logic in Computer Science (LICS '99)*, pp. 295–305, Trento, Italy. IEEE Computer Society.

Georgieva, L., Hustadt, U., & Schmidt, R. A. (2003). Hyperresolution for Guarded Formulae. *Journal of Symbolic Computation*, *36*(1–2), 163–192.

Glimm, B., Horrocks, I., Motik, B., Shearer, R., & Stoilos, G. (2012). A Novel Approach to Ontology Classification. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, *14*, 84–101.

Glimm, B., Horrocks, I., Motik, B., Stoilos, G., & Wang, Z. (2014). HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning*, *53*(3), 245–269.

Goré, R., & Nguyen, L. A. (2013). ExpTime Tableaux for $\mathcal{ALC}$ Using Sound Global Caching. *Journal of Automated Reasoning*, *50*(4), 355–381.

Goré, R., & Nguyen, L. A. (2007). EXPTIME Tableaux with Global Caching for Description Logics with Transitive Roles, Inverse Roles and Role Hierarchies. In Olivetti, N. (Ed.), *Proc. of the 16th Int. Conf. on Automated Reasoning with Tableaux and Related Methods (TABLEAUX 2007)*, Vol. 4548 of *LNCS*, pp. 133–148, Aix en Provence, France. Springer.

Graf, P. (1996). *Term Indexing*, Vol. 1053 of *LNCS*. Springer.

Green, C. (1969). Theorem proving by resolution as a basis for question-answering systems. In Meltzer, B., & Michie, D. (Eds.), *Proc. of the 4th Annual Machine Intelligence Workshop*, pp. 183–208. Edinburgh University Press.

Haarslev, V., Timmann, M., & Möller, R. (2001). Combining Tableaux and Algebraic Methods for Reasoning with Qualified Number Restrictions. In Goble, C., Möller, R., & Patel-Schneider, P. F. (Eds.), *Proc. of the 2001 Int. Workshop on Description Logics (DL 2001)*, Vol. 49 of *CEUR Workshop Proceedings*, Stanford, CA, USA.

Haarslev, V., Hidde, K., Möller, R., & Wessel, M. (2012). The RacerPro knowledge representation and reasoning system. *Semantic Web*, *3*(3), 267–277.

Horridge, M., & Bechhofer, S. (2011). The OWL API: A Java API for OWL ontologies. *Semantic Web*, *2*(1), 11–21.

Horrocks, I., & Sattler, U. (2004). Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. *Artificial Intelligence*, *160*(1–2), 79–104.

Hustadt, U., & Schmidt, R. A. (1999). Issues of Decidability for Description Logics in the Framework of Resolution. In Caferra, R., & Salzer, G. (Eds.), *Selected Papers from Automated Deduction in Classical and Non-Classical Logics*, Vol. 1761 of *LNAI*, pp. 191–205. Springer.

Hustadt, U., Motik, B., & Sattler, U. (2007). Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *Journal of Automated Reasoning*, *39*(3), 351–384.

Hustadt, U., Motik, B., & Sattler, U. (2008). Deciding Expressive Description Logics in the Framework of Resolution. *Information & Computation*, *206*(5), 579–601.

Hustadt, U., & Schmidt, R. A. (2002). Using Resolution for Testing Modal Satisfiability and Building Models. *Journal of Automated Reasoning*, *28*(2), 205–232.

Karahroodi, N. Z., & Haarslev, V. (2017). A Consequence-based Algebraic Calculus for $\mathcal{SHOQ}$. In Artale, A., Glimm, B., & Kontchakov, R. (Eds.), *Proc. of the 30th Int. Workshop on Description Logics (DL 2017)*, Vol. 1879 of *CEUR Workshop Proceedings*, Montpellier, France.

Kazakov, Y. (2008). $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are Harder than $\mathcal{SHOIQ}$. In Brewka, G., & Lang, J. (Eds.), *Proc. of the 11th Int. Joint Conf. on Principles of Knowledge Representation and Reasoning (KR 2008)*, pp. 274–284, Sydney, NSW, Australia. AAAI Press.

Kazakov, Y. (2009). Consequence-Driven Reasoning for Horn $\mathcal{SHIQ}$ Ontologies. In Boutilier, C. (Ed.), *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009)*, pp. 2040–2045, Pasadena, CA, USA.

Kazakov, Y., & Klinov, P. (2014). Bridging the Gap between Tableau and Consequence-Based Reasoning. In Bienvenu, M., Ortiz, M., Rosati, R., & Simkus, M. (Eds.), *Proc.*

of the 27th Int. Workshop on Description Logics (DL 2014), Vol. 1193 of *CEUR Workshop Proceedings*, pp. 579–590, Vienna, Austria.

Kazakov, Y., Krötzsch, M., & Simančík, F. (2011). Concurrent Classification of $\mathcal{EL}$ Ontologies. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N. F., & Blomqvist, E. (Eds.), *Proc. of the 10th Int. Semantic Web Conference (ISWC 2011)*, Vol. 7031, pp. 305–320, Bonn, Germany. Springer.

Kazakov, Y., Krötzsch, M., & Simančík, F. (2014). The Incredible ELK: From Polynomial Procedures to Efficient Reasoning with $\mathcal{EL}$ Ontologies. *Journal of Automated Reasoning*, *53*(1), 1–61.

Knublauch, H., Fergerson, R. W., Noy, N. F., & Musen, M. A. (2004). The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In McIlraith, S. A., Plexousakis, D., & van Harmelen, F. (Eds.), *Proc. of the 3rd Int. Semantic Web Conference (ISWC 2004)*, Vol. 3298 of *LNCS*, pp. 229–243, Hiroshima, Japan. Springer.

McCune, W., & Wos, L. (1997). Otter—The CADE-13 Competition Incarnations. *Journal of Automated Reasoning*, *18*(2), 211–220.

Mendez, J. (2012). jcel: A Modular Rule-based Reasoner. In Horrocks, I., Yatskevich, M., & Jiménez-Ruiz, E. (Eds.), *Proc. of the 1st Int. Workshop on OWL Reasoner Evaluation (ORE-2012)*, Vol. 858 of *CEUR Workshop Proceedings*, Manchester, UK.

Metke-Jimenez, A., & Lawley, M. (2013). Snorocket 2.0: Concrete Domains and Concurrent Classification. In Bail, S., Glimm, B., Gonçalves, R. S., Jiménez-Ruiz, E., Kazakov, Y., Matentzoglu, N., & Parsia, B. (Eds.), *Proc. of the 2nd Int. Workshop on OWL Reasoner Evaluation (ORE-2013)*, Vol. 1015 of *CEUR Workshop Proceedings*, pp. 32–38, Ulm, Germany.

Motik, B., Shearer, R., & Horrocks, I. (2009). Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, *36*, 165–228.

Nguyen, L. A. (2013). A Tableau Method with Optimal Complexity for Deciding the Description Logic SHIQ. In Nguyen, N. T., Do, T. V., & Thi, H. A. L. (Eds.), *Advanced Computational Methods for Knowledge Engineering*, Vol. 479 of *SCI*, pp. 331–342. Springer.

Nguyen, L. A. (2014). ExpTime tableaux with global state caching for the description logic SHIO. *Neurocomputing*, *146*, 249–263.

Nguyen, L. A., & Golińska-Pilarek, J. (2014). An ExpTime Tableau Method for Dealing with Nominals and Qualified Number Restrictions in Deciding the Description Logic SHOQ. *Fundamenta Informaticae*, *135*(4), 433–449.

Nieuwenhuis, R., & Rubio, A. (1995). Theorem Proving with Ordering and Equality Constrained Clauses. *Journal of Symbolic Computation*, *19*(4), 312–351.

Nieuwenhuis, R., & Rubio, A. (2001). Paramodulation-Based Theorem Proving. In Robinson, A., & Voronkov, A. (Eds.), *Handbook of Automated Reasoning*, Vol. I, chap. 7, pp. 371–443. Elsevier Science.

Nonnengart, A., & Weidenbach, C. (2001). Computing Small Clause Normal Forms. In Robinson, A., & Voronkov, A. (Eds.), *Handbook of Automated Reasoning*, Vol. I, chap. 6, pp. 335–367. Elsevier Science.

Ramakrishnan, I. V., Sekar, R., & Voronkov, A. (2001). Term Indexing. In Robinson, A., & Voronkov, A. (Eds.), *Handbook of Automated Reasoning*, Vol. II, chap. 26, pp. 1853–1964. Elsevier Science.

Riazanov, A., & Voronkov, A. (2002). The design and implementation of VAMPIRE. *AI Communications*, *15*(2–3), 91–110.

Riazanov, A., & Voronkov, A. (2003). Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computation*, *36*(1-2), 101–115.

Sattler, U., & Vardi, M. Y. (2001). The Hybrid μ-Calculus. In Goré, R., Leitsch, A., & Nipkow, T. (Eds.), *Proc. of the 1st Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, Vol. 2083 of *LNCS*, pp. 76–91, Siena, Italy. Springer.

Schmidt, R. A., & Hustadt, U. (2007). The Axiomatic Translation Principle for Modal Logic. *ACM Transactions on Computational Logic*, *8*(4).

Schmidt, R. A., & Hustadt, U. (2013). First-Order Resolution Methods for Modal Logics. In Voronkov, A., & Weidenbach, C. (Eds.), *Programming Logics—Essays in Memory of Harald Ganzinger*, Vol. 7797 of *LNCS*, pp. 345–391. Springer.

Schulz, S. (2002). E—A Brainiac Theorem Prover. *AI Communications*, *15*(2–3), 111–126.

Schulz, S. (2004). Simple and Efficient Clause Subsumption with Feature Vector Indexing. In Schulz, S., Sutcliffe, G., & Tammet, T. (Eds.), *The IJCAR 2004 Workshop on Empirically Successful First Order Reasoning (ESFOR)*, Cork, Ireland.

Schulz, S. (2013). System Description: E 1.8. In McMillan, K. L., Middeldorp, A., & Voronkov, A. (Eds.), *Proc. of the 19th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-19)*, Vol. 8312 of *LNCS*, pp. 735–743, Stellenbosch, South Africa. Springer.

Sertkaya, B. (2011). In the Search of Improvements to the $\mathcal{EL}^+$ Classification Algorithm. In Rosati, R., Rudolph, S., & Zakharyaschev, M. (Eds.), *Proc. of the 24th Int. Workshop on Description Logics (DL 2011)*, Vol. 745 of *CEUR Workshop Proceedings*, Barcelona, Spain.

Simančík, F. (2012). Elimination of complex RIAs without automata. In Kazakov, Y., Lembo, D., & Wolter, F. (Eds.), *Proc. of the 2012 Int. Workshop on Description Logics (DL 2012)*, Vol. 846 of *CEUR Workshop Proceedings*, Rome, Italy.

Simančík, F., Motik, B., & Horrocks, I. (2014). Consequence-Based and Fixed-Parameter Tractable Reasoning in Description Logics. *Artificial Intelligence*, *209*, 29–77.

Simančík, F., Kazakov, Y., & Horrocks, I. (2011). Consequence-Based Reasoning beyond Horn Ontologies. In Walsh, T. (Ed.), *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011)*, pp. 1093–1098, Barcelona, Spain.

Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, *5*(2), 51–53.

Solomon, W., Roberts, A., Rogers, J. E., C.J., C. J. W., & Rector, A. L. (2000). Having our cake and eating it too: How the GALEN Intermediate Representation reconciles internal complexity with users' requirements for appropriateness and simplicity. In *Proc. of the Annual Fall Symposium of American Medical Informatics Association*, pp. 819–823, Los Angeles, CA, USA.

Steigmiller, A., Liebig, T., & Glimm, B. (2014). Konclude: System description. *Journal of Web Semantics*, *27*, 78–85.

Tobies, S. (2001). *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. Ph.D. thesis, RWTH Aachen, Germany.

Tsarkov, D., & Horrocks, I. (2006). FaCT++ Description Logic Reasoner: System Description. In *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, Vol. 4130 of *LNAI*, pp. 292–297, Seattle, WA, USA. Springer.

Vlasenko, J., Daryalal, M., Haarslev, V., & Jaumard, B. (2016). A Saturation-based Algebraic Reasoner for $\mathcal{ELQ}$. In Fontaine, P., Schulz, S., & Urban, J. (Eds.), *Proc. of the 5th Workshop on Practical Aspects of Automated (PAAR 2016)*, Vol. 1635 of *CEUR Workshop Proceedings*, pp. 110–124, Coimbra, Portugal.

Walther, C. (1987). *A Many-Sorted Calculus Based on Resolution and Paramodulation*. Morgan Kaufmann.

Weidenbach, C., Gaede, B., & Rock, G. (1996). SPASS & FLOTTER Version 0.42. In McRobbie, M. A., & Slaney, J. K. (Eds.), *Proc. of the 13th Int. Conf. on Automated Deduction (CADE-13)*, Vol. 1104 of *LNCS*, pp. 141–145, New Brunswick, NJ, USA. Springer.