Extending Consequence-Based Reasoning to $SRIQ$

Andrew Bate, Boris Motik, Bernardo Cuenca Grau, František Simančík, Ian Horrocks
Department of Computer Science, University of Oxford, Oxford, United Kingdom
firstname.lastname@cs.ox.ac.uk

Abstract
Consequence-based calculi are a family of reasoning algorithms for description logics (DLs), and they combine hyper-tableau and resolution in a way that often achieves excellent performance in practice. Up to now, however, they were proposed for either Horn DLs (which do not support disjunction), or for DLs without counting quantifiers. In this paper we present a novel consequence-based calculus for $SRIQ$—a rich DL that supports both features. This extension is non-trivial since the intermediate consequences that need to be derived during reasoning cannot be captured using DLs themselves. The results of our preliminary performance evaluation suggest the feasibility of our approach in practice.

1 Introduction

Description logics (DLs) (Baader et al. 2003) are a family of knowledge representation formalisms with numerous applications in practice. DL-based applications model a domain of interest by means of an ontology, in which key notions in the domain are described using concepts (i.e., unary predicates), and the relationships between concepts are described using roles (i.e., binary predicates). Subsumption is the problem of determining whether each instance of a concept $C$ is also an instance of a concept $D$ in all models of an ontology, and it is a fundamental reasoning problem in applications of DLs. For expressive DLs, this problem is of high worst-case complexity, ranging from EXPTIME up to $N2EXPTIME$.

Despite these discouraging complexity bounds, highly optimised reasoners such as FaCT++ (Tsarkov and Horrocks 2006), Pellet (Sirin et al. 2007), HermiT (Glimm et al. 2014), and Konclude (Steigmiller, Liebig, and Glimm 2014) have proved successful in practice. These systems are typically based on (hyper)tableau calculi, which construct a finite representation of a canonical model of the ontology disproving a postulated subsumption. While such calculi can handle many ontologies, in some cases they construct very large model representations, which is a source of performance problems; this is further exacerbated by the large number of subsumption tests often required to classify an ontology.

A recent breakthrough in DL reasoning came in the form of consequence-based calculi. The reasoning algorithm by Baader, Brandt, and Lutz (2005) for the lightweight logic $\mathcal{EL}$ can be seen as the first such calculus. It was later extended to the more expressive DLs Horn-$\mathcal{SHIQ}$ (Kazakov 2009) and Horn-$\mathcal{SROIQ}$ (Ortiz, Rudolph, and Simkus 2010)—DLs that support counting quantifiers, but not disjunctions between concepts. Consequence-based calculi were also developed for $\mathcal{ALCH}$ (Simančík, Kazakov, and Horrocks 2011) and $\mathcal{ALCI}$ (Simančík, Motik, and Horrocks 2014), which support concept disjunction, but not counting quantifiers. Such calculi can be seen as combining resolution and hyper-tableau (see Section 3 for details); as in resolution, they describe ontology models by systematically deriving relevant consequences; and as in (hyper)tableau, they are goal-directed and avoid drawing unnecessary consequences. Additionally, they are not only refutationally complete, but can also (dis)prove all relevant subsumptions in a single run, which can greatly reduce the overall computational work. Finally, unlike implemented (hyper)tableau reasoners, they are worst-case optimal for the logic they support. Steigmiller, Glimm, and Liebig (2014) presented a way of combining a consequence-based calculus with a traditional tableau-based prover; while such a combination seems to perform well in practice, the saturation rules are only known to be complete for $\mathcal{EL}$ ontologies, and the overall approach is not worst-case optimal for $SRIQ$.

Existing consequence-based algorithms cannot handle DLs such as $\mathcal{ALCHIQ}$ that provide both disjunctions and counting quantifiers. As we argue in Section 3, extending these algorithms to handle such DLs is challenging: counting quantifiers require equality reasoning which, together with disjunctions, can impose complex constraints on ontology models; and, unlike existing consequence-based calculi, such constraints cannot be captured using DLs themselves, which makes the reasoning process much more involved.

In Section 4 we present a consequence-based calculus for $\mathcal{ALCHIQ}$; by using the encoding of role chains by Kazakov (2008), our calculus can also handle $SRIQ$, which covers all of OWL 2 DL except for nominals, reflexive roles, and datatypes. Borrowing ideas from resolution theorem proving, we encode the calculus’ consequences as first-order clauses of a specific form, and we handle equality using a variant of ordered paramodulation (Nieuwenhuis and Rubio 1995)—a state of the art calculus for equational theorem proving used in modern theorem provers such as E (Schulz 2002) and Vampire (Riazanov and Voronkov 2002). Furthermore, we have carefully constrained the inference rules so that our calculus mimics existing calculi on $\mathcal{ELH}$ ontolo-
eties, which ensures robust performance of our calculus on ‘mostly-ELH’ ontologies.

We have implemented a prototype system and compared its performance with that of well-established reasoners. Our results in Section 5 suggest that our system can significantly outperform FaCT++, Pellet, or HermiT, and often exhibits comparable performance to that of Konclude.

2 Preliminaries

First-Order Logic. It is usual in equational theorem proving to encode atomic formulas as terms, and to use a multi-sorted signature that prevents us from considering malformed terms. Thus, we partition the signature into a set \( \mathcal{P} \) of predicate symbols and a set \( \mathcal{F} \) of function symbols; moreover, we assume that \( \mathcal{P} \) has a special constant \( \varnothing \). A term is constructed as usual using variables and the signature symbols, with the restriction that predicate symbols are allowed to occur only at the outermost level; the latter terms are called \( \mathcal{P}\text{-}terms \) while all other terms are \( \mathcal{F}\text{-}terms \). For example, for \( P \) a predicate and \( f \) a function symbol, \( f(P(x)) \) and \( P(f(x)) \) are both malformed; \( P(f(x)) \) is a well-formed \( \mathcal{P}\text{-}term \) and \( f(x) \) and \( x \) are both well-formed \( \mathcal{F}\text{-}terms \). Term \( f(t) \) is an \( f\text{-}successor \) of \( t \), and \( t \) is an \( f\text{-}predecessor \) of \( f(t) \).

An equality is a formula of the form \( s \approx t \), where \( s \) and \( t \) are either both \( \mathcal{F}\text{-} \) or both \( \mathcal{P}\text{-}terms \). An equality of the form \( P(s) \approx \varnothing \) is called an atom and is written as just \( P(s) \) whenever it is clear from the context that the expression denotes a formula, and not a \( \mathcal{P}\text{-}term \). An inequality is a negation of an equality and is written as \( s \neq t \). We assume that \( \approx \) and \( \neq \) are implicitly symmetric—that is, \( s \approx t \) and \( t \approx s \) are identical, for \( \approx \in \{\approx, \neq\} \). A literal is an equality or an inequality. A clause is a formula of the form \( \forall \vec{x} [\Gamma \rightarrow \Delta] \) where \( \Gamma \) is a conjunction of atoms called the body, \( \Delta \) is a disjunction of literals called the head, and \( \vec{x} \) contains all variables occurring in the clause; quantifier \( \forall \vec{x} \) is usually omitted as it is understood implicitly. We often treat conjunctions and disjunctions as sets (i.e., they are unordered and without repetition) and use them in standard set operations; and we write the empty conjunction (disjunction) as \( \top \) (\( \bot \)). For \( \alpha \) a term, literal, clause, or a set thereof, we say that \( \alpha \) is ground if it does not contain a variable; \( \alpha \sigma \) is the result of applying a substitution \( \sigma \) to \( \alpha \); and we often write substitutions as \( \sigma = \{x \mapsto t_1, y \mapsto t_2, \ldots\} \). We use the standard notion of subterm positions; \( s|p \) is the subterm of \( s \) at position \( p \); position \( p \) is proper in a term \( t \) if \( t|p \neq t \); and \( s|t|p \) is the term obtained by replacing the subterm of \( s \) at position \( p \) with \( t \).

A Herbrand equality interpretation is a set of ground equalities satisfying the usual congruence properties. Satisfaction of a ground conjunction, a ground disjunction, or a (not necessarily ground) clause \( \alpha \) in an interpretation \( I \), written \( I \models \alpha \), as well as entailment of a clause \( \Gamma \rightarrow \Delta \) from a set of clauses \( \mathcal{O} \), written \( \mathcal{O} \models \Gamma \rightarrow \Delta \), are defined as usual. Note that a ground disjunction of literals \( \Delta \) may contain inequalities so \( I \models \Delta \) does not necessarily imply \( I \cap \Delta \neq \emptyset \).

Orders. A strict order \( \succ \) on a universe \( U \) is an irreflexive, asymmetric, and transitive relation on \( U \); and \( \succeq \) is the non-strict order induced by \( \succ \). Order \( \sim \) is total if, for all \( a, b \in U \), we have \( a \succ b \), \( b \succ a \), or \( a = b \). Given \( o \in \{\succ, \succeq\} \), element \( b \in U \), and subset \( S \subseteq U \), the notation \( S \circ b \) abbreviates \( \{ a \in S \mid a \sim b \} \). The multiset extension \( \succ_{\text{mul}} \) of \( \succ \) compares multisets \( M \) and \( N \) on \( U \) such that \( M \succ_{\text{mul}} N \) if and only if \( M \neq N \) and, for each \( n \in N \setminus M \), some \( m \in M \setminus N \) exists such that \( m \succ n \), where \( \setminus \) is the multiset difference operator.

A term order \( \succ \) is a strict order on the set of all terms. We extend \( \succ \) to literals by identifying each \( s \not\succ t \) with the multiset \( \{s, t\} \) and each \( s \sim t \) 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.

DL-Clauses. Our calculus takes as input a set \( \mathcal{O} \) of DL-clauses—that is, clauses restricted to the following form. Let \( \mathcal{P}_1 \) and \( \mathcal{P}_2 \) be countable sets of unary and binary predicate symbols, and let \( \mathcal{F} \) be a countable set of unary function symbols. DL-clauses are written using the central variable \( x \) and variables \( z_i \). A DL-F-term has the form \( x, z_i \) or \( f(x) \) with \( f \in \mathcal{F} \); a DL-P-term has the form \( B(z_i), B(f(x)), S(x, z_i), S(z_i, x), S(x, f(x)), S(f(x), x) \) with \( B \in \mathcal{P}_1 \) and \( S \in \mathcal{P}_2 \); and a DL-term is a DL-F-term or a DL-P-term. A DL-atom has the form \( A \approx \varnothing \) with \( A \) a DL-P-term. A DL-literal is a DL-atom, or it is of the form \( f(x) \approx g(x) \), \( f(x) \approx z_i \), \( z_i \approx z_j \) with \( \approx \in \{\approx, \neq\} \). A DL-clause contains only DL-atoms of the form \( B(x), S(x, z_i), S(z_i, x) \) in the body and only DL-literals in the head, and each variable \( z_i \) occurring in the head also occurs in the body. An ontology \( \mathcal{O} \) is a finite set of DL-clauses. A query clause is a DL-clause in which all literals are of the form \( B(x) \). Given an ontology \( \mathcal{O} \) and a query clause \( \Gamma \rightarrow \Delta \), our calculus decides whether \( \mathcal{O} \models \Gamma \rightarrow \Delta \) holds.

SRIQ ontologies written using the DL-style syntax can be transformed into DL-clauses without affecting query clause entailment. First, we normalise DL axioms to the form shown on the left-hand side of Table 1: we transform away role chains and then replace all complex concepts with fresh atomic ones; this process is well understood (Kazakov 2009; 2008; Simančík, Motik, and Horrocks 2014), so we omit the details. Second, using the well-known correspondence between DLs and first-order logic (Baader et al. 2003), we translate normalised axioms to DL-clauses as shown on the right-hand side of Table 1. The standard translation of \( B_1 \subseteq n S.B_2(z_i) \) requires atoms \( B_2(z_i) \) in clause bodies, which are not allowed in our setting. We address this issue by introducing a fresh role \( S.B_2 \) that we axiomatise as \( S(y, x) \land B_2(x) \rightarrow S.B_2(y, x) \); this, in turn, allows us to classify the original axiom as if it were \( B_1 \subseteq S.B_2 \). For an ELH ontology, \( \mathcal{O} \) contains DL-clauses of type DL1 with \( m = n + 1 \), DL2 with \( n = 1 \), DL3, and DL5.

3 Motivation

As motivation for our work, in Section 3.1 we discuss the drawbacks of existing DL reasoning calculi, and then in Section 3.2 we discuss how existing consequence-based calculi
niques such as (hyper)tableau calculi from being worst-case optimal. Tech-
apart from causing problems in practice, this often prevents
be large (exponential in our example) and highly redundant;
and they share the same properties, which reveals a weak-
B
Forward chaining ensures that reasoning is goal-oriented;
constructs a tree-shaped model of depth
n
B
prove
B
 ontology
are of the same type
are relevant to
all (4) to obtain all
n
Bn
Bn
Cn

Table 1: Translating Normalised ALCHIQ Ontologies into DL-Clauses

| DL | \[ \prod_{1 \leq i \leq n} B_i \cup \bigcup_{n+1 \leq i \leq m} B_i \] \implies \bigwedge_{1 \leq i \leq n} B_i(x) \rightarrow \bigvee_{n+1 \leq i \leq m} B_i(x) |
| DL2 | B_1 \cup \exists n S.B_2 \implies B_1(x) \rightarrow S(x, f_i(x)) \quad \text{for } 1 \leq i \leq n |
| DL3 | \exists S.B_1 \subseteq B_2 \implies S(z_1, x) \wedge B_1(x) \rightarrow B_2(z_1) |
| DL4 | B_1 \subseteq \exists n S.B_2 \implies B_1(x) \wedge \bigwedge_{1 \leq i \leq n+1} S_{B_2}(x, z_i) \rightarrow \bigvee_{1 \leq i \leq j \leq n+1} z_i \approx z_j |
| DL5 | S_1 \subseteq S_2 \implies S_1(z_1, x) \rightarrow S_2(x, z_1) |
| DL6 | S_1 \subseteq S_2 \implies S_1(z_1, x) \rightarrow S_2(x, z_1) |

| Ontology \( \mathcal{O}_1 \) |
| \[ B_i \sqsubseteq \exists S_j B_{i+1} \] \implies B_i(x) \rightarrow S_j(x, f_{i+1,j}(x)) \quad \text{(1)} \quad \text{for } 0 \leq i < n \text{ and } 1 \leq j \leq 2 |
| \[ B_n \sqsubseteq C_n \] \implies B_n(x) \rightarrow C_n(x) \quad \text{(3)} |
| \[ \exists S_j C_{i+1} \subseteq C_i \] \implies S_j(z_1, x) \wedge C_{i+1}(x) \rightarrow C_i(z_1) \quad \text{(4)} \quad \text{for } 0 \leq i < n \text{ and } 1 \leq j \leq 2 |

address these problems by separating clauses into contexts in a way that considerably reduces the number of infer-
cees. Next, in Section 3.3 we discuss the main contribution of
this paper, which lies in extending the consequence-based
framework to a DL with disjunctions and number restric-
tions. Handling the latter requires equality reasoning, which
requires a more involved calculus and completeness proof.

3.1 Why Consequence-Based Calculi?

Consider the \( \mathcal{EL} \) ontology \( \mathcal{O}_1 \) in Figure 1; one can read-
ily check that \( \mathcal{O} \models B_i(x) \rightarrow C_i(x) \) holds for \( 0 \leq i \leq n \). To
prove \( \mathcal{O} \models B_0(x) \rightarrow C_0(x) \) using the (hyper)tableau cal-
culus, we start with \( B_0(a) \) and apply (1)–(4) in a forward-
chaining manner. Since \( \mathcal{O} \) contains (1) for \( j \in \{1, 2\} \), this
constructs a tree-shaped model of depth \( n \) and a fanout of
two, where nodes at depth \( i \) are labelled by \( B_i \) and \( C_i \).
Forward chaining ensures that reasoning is goal-oriented;
however, all nodes labelled with \( B_i \) are of the same type
and they share the same properties, which reveals a weak-
ness of (hyper)tableau calculi: the constructed models can
be large (exponential in our example) and highly redundant;
 apart from causing problems in practice, this often prevents
(hyper)tableau calculi from being worst-case optimal. Tech-
niques such as caching (Goré and Nguyen 2007) or any-
where blocking (Motik, Shearer, and Horrocks 2009) can
constrain model construction, but their effectiveness often
depends on the order of rule applications. Thus, model size
is a key limiting factor for (hyper)tableau-based reasoners
(Motik, Shearer, and Horrocks 2009).

In contrast, resolution describes models using (univer-
sally quantified) clauses that ‘summarise’ the model. This
eliminates redundancy and ensures worst-case optimality
of many resolution decision procedures. Many resolution vari-
ants have been proposed (Bachmair and Ganzinger 2001),
each restricting inferences in a specific way. However, to
ensure termination, all decision procedure for DLs we are
aware of perform inferences with the ‘deepest’ and the ‘cov-
ering’ clause atoms, so all of them will resolve all (1) with
all (4) to obtain all \( 2n^2 \) clauses of the form

\[
B_i(x) \wedge C_{k+1}(f_{i+1,j}(x)) \rightarrow C_k(x) \quad \text{for } 1 \leq i < k < n \text{ and } 1 \leq j \leq 2.
\]  

(21)

Of these \( 2n^2 \) clauses, only those with \( i = k \) are relevant to
proving our goal. If we extend \( \mathcal{O} \) with additional clauses that
contain \( B_i \) and \( C_i \), each of these \( 2n^2 \) clauses can participate
in further inferences and give rise to more irrelevant clauses.
This problem is particularly pronounced when \( \mathcal{O} \) is satisfi-
able since we must then produce all consequences of \( \mathcal{O} \).
3.2 Basic Notions

Consequence-based calculi combine ‘summarisation’ of resolution with goal-directed search of (hyper)tableau calculi. Simančík, Motik, and Horrocks (2014) presented a framework for ALCII capturing the key elements of the related calculi by Baader, Brandt, and Lutz (2005), Kazakov (2009), Ortiz, Rudolph, and Simkus (2010), and Simančík, Kazakov, and Horrocks (2011). Before extending this framework to ACHIQ in Section 4, we now informally recapitulate the basic notions; however, to make this paper easier to follow, we use the same notation and terminology as in Section 4.

Our consequence-based calculus constructs a directed graph \( D = (V, E, S, \text{core}, \succ) \) called a context structure. The vertices in \( V \) are called contexts. Let \( I \) be a Herbrand model of \( \mathcal{O} \); hence, the domain of \( I \) contains ground terms. Instead of representing each ground term of \( I \) separately as in (hyper)tableau calculi, \( D \) can represent the properties of several terms by a single context \( v \). Each context \( v \in V \) is associated with a (possibly empty) conjunction \( \text{core}_v \) of core atoms that must hold for all ground terms that \( v \) represents; thus, \( \text{core}_v \) determines the ‘kind’ of context \( v \). Moreover, \( v \) is associated with a set \( S_v \) of clauses that capture the constraints that these terms must satisfy. Partitioning clauses into sets allows us to restrict the inferences between clause sets and thus eliminate certain irrelevant inferences. Clauses in \( S_v \) are ‘relative’ to \( \text{core}_v \); for each \( \Gamma \models \text{core}_v \wedge \Delta \rightarrow \Delta \) that is, we do not include \( \text{core}_v \) in clause bodies since \( \text{core}_v \) holds implicitly. Function \( \succ \) provides each context \( v \in V \) with a concept order \( \succ_v \), that restricts resolution inferences in the presence of disjunctions.

Contexts are connected by directed edges labelled with function symbols. If \( u \) is connected to \( v \) via an \( f \)-labelled edge, then the \( f \)-successor of each ground term represented by \( u \) is represented by \( v \). Conversely, if \( u \) and \( v \) are not connected by an \( f \)-edge, then each ground term represented by \( v \) is not an \( f \)-successor of a ground term represented by \( u \), so no inference between \( S_u \) and \( S_v \) is ever needed.

Consequence-based calculi are not just complete for refutation: they derive the required consequences. Figure 1 demonstrates this for \( O_1 \models B_0(x) \rightarrow C_0(x) \). The cores and the clauses shown above and below, respectively, each context, and clause numbers correspond to the derivation order. To prove \( B_0(x) \rightarrow C_0(x) \), we introduce context \( v_{B_0(x)} \) with core \( B_0(x) \) and add clause (5) to it. The latter says that \( B_0 \) holds for \( a \), and it is analogous to initialising a (hyper)tableau calculus with \( B_0(a) \). The calculus then applies rules from Table 2 to derive new clauses and/or extend \( D \).

Hyper is the standard hyperresolution rule restricted to a single context at a time. Thus, we derive (6) from (1) and (5), and (7) from (2) and (5). Hyperresolution resolves all body atoms, which makes the resolvent relevant for the context and prevents the derivation of irrelevant clauses such as (21).

Context \( v_{B_0(x)} \) contains atoms with function symbols \( f_{1,1} \) and \( f_{1,2} \), so the Succ rule must ensure that the \( f_{1,1} \) and \( f_{1,2} \)-successors of the ground terms represented by \( v_{B_0(x)} \) are adequately represented in \( D \). We can control context introduction via a parameter called an expansion strategy—a function that determines whether to reuse an existing context or introduce a fresh one; in the latter case, it also determines how to initialise the context’s core. We discuss possible strategies in Section 4.1; in the rest of this example, we use the so-called cautious strategy, where the Succ rule introduces context \( v_{B_1(x)} \) and initialises it with (10) and (11). Note that (6) represents two clauses, both of which we satisfy (in separate applications of the Succ rule) using \( v_{B_1(x)} \).

We construct contexts \( v_{B_0(x)}, \ldots, v_{B_4(x)} \) analogously, we derive (16) by hyperresolving (3) and (14), and we derive (17) by hyperresolving (4), (14), and (16). Clause (17) imposes a constraint on the predecessor context, which we propagate using the Pred rule, deriving (19) and (20). Since clauses of \( v_{B_4(x)} \) are ‘relative’ to the core of \( v_{B_0(x)} \), Clause (20) represents our query clause, as required.

3.3 Extending the Framework to ACHIQ

In all consequence-based calculi presented thus far, the constraints that the ground terms represented by a context \( v \) must satisfy can be represented using standard DL-style axioms. For example, for ALCII, Simančík, Motik, and Horrocks (2014) represented all relevant consequences using DL axioms of the following form:

\[
\bigcap B_i \subseteq \bigcup B_j \cup \exists S_k.B_k \cup \forall S_I.B_I \quad (55)
\]

ACHIQ provides both counting quantifiers and disjunctions, the interplay of which may impose constraints that cannot be represented in ACHIQ. Let \( O_2 \) be as in Figure 2. To see that \( O_2 \models B_0(x) \rightarrow B_4(x) \), we construct a Herbrand interpretation \( I \) from \( B_0(a) \): (22) and (23) derive \( S(f_1(a), a) \) and \( B_1(f_1(a)) \); and (25) and (26) derive \( S(f_1(a), f_2(f_1(a))) \) and \( B_2(f_2(f_1(a))) \), and \( S(f_1(a), f_3(f_1(a))) \) and \( B_3(f_3(f_1(a))) \). Due to (27) we derive \( B_4(f_2(f_1(a))) \) and \( B_4(f_3(f_1(a))) \). Finally, from (28) we derive the following clause:

\[
f_2(f_1(a)) \approx a \vee f_3(f_1(a)) \approx a \vee f_2(f_1(a)) \approx f_2(f_1(a)) \quad (56)
\]

Disjunct \( f_3(f_1(a)) \approx f_2(f_1(a)) \) cannot be satisfied due to (24); but then, regardless of whether we choose to satisfy \( f_3(f_1(a)) \approx a \) or \( f_2(f_1(a)) \approx a \), we derive \( B_4(a) \).

Our calculus must be able to capture constraint (56) and its consequences, but standard DL axioms cannot explicitly refer to specific successors and predecessors. Instead, we capture consequences using context clauses—clauses over terms \( x, f_i(x), \) and \( y \), where variable \( x \) represents the ground terms that a context stands for, \( f_i(x) \) represents \( f_i \)-successors of \( x \), and \( y \) represents the predecessor of \( x \). We can thus identify the predecessor and the successors of \( x \) ‘by name’, allowing us to capture constraint (56) as

\[
f_2(x) \approx y \vee f_3(x) \approx y \vee f_3(x) \approx f_2(x). \quad (57)
\]

Based on this idea, we adapted the rules by Simančík, Motik, and Horrocks (2014) to handle context clauses correctly, and we added rules that capture the consequences of equality. The resulting set of rules is shown in Table 2.

Figure 2 shows how to verify \( O_2 \models B_0(x) \rightarrow B_4(x) \) using our calculus; the maximal literal of each clause is shown on the right. We next discuss the inferences in detail.
Ontology $O_2$

$$B_0 \subseteq \exists S^-.B_1 \quad \leadsto \quad B_0(x) \rightarrow S(f_1(x), x) \quad (22)$$

$$B_0(x) \rightarrow B_1(f_1(x)) \quad (23)$$

$$B_1 \subseteq \exists S.B_1 \quad \leadsto \quad B_1(x) \rightarrow B_2(f_1(x)) \quad (26)$$

$$B_1(x) \rightarrow B_2(f_1(x)) \quad (25)$$

$$B_2 \subseteq B_4 \quad \leadsto \quad B_4(x) \rightarrow B_3(x) \quad (27)$$

for $2 \leq i \leq 3$

$B_2 \cap B_3 \subseteq \bot \quad \leadsto \quad B_2(x) \land B_3(x) \rightarrow \bot \quad (24)$

$B_1 \subseteq 2.S \quad \leadsto \quad B_1(x) \land \bigwedge_{1 \leq i \leq 3} S(x, z_i) \rightarrow \bigvee_{1 \leq j \leq k \leq 3} z_j \approx z_k \quad (28)$

Figure 2: Challenges in Extending the Consequence-Based Framework to ALCHIQ

We first create context $v_0$ and initialise it with (29); this ensures that each interpretation represented by the context structure contains a ground term for which $B_0$ holds. Next, we derive (30) and (31) using hyperresolution. At this point, we could hyperresolve (25) and (31) to obtain $T \rightarrow S(f_1(x), f_2(f_1(x)))$; however, this would easily lead to nontermination of the calculus due to increased term nesting. Therefore, we require hyperresolution to map variable $x$ in the DL-clauses to variable $z$ in the context clauses; thus, hyperresolution derives in each context only consequences about $z$, which prevents redundant derivations.

The Succ rule next handles function symbol $f_1$ in clauses (30) and (31). To determine which information to propagate to a successor, Definition 2 in Section 4 introduces a set $SU(O)$ of successor triggers. In our example, DL-clause (28) contains atoms $B_1(x)$ and $S(x, z_i)$ in its body, and $z_i$ can be mapped to a predecessor or a successor of $x$; thus, a context in which hyperresolution is applied to (28) will be interested in information about its predecessors, which we reflect by adding $B_1(x)$ and $S(x, y)$ to $SU(O)$. In this example we use the so-called eager strategy (see Section 4.1), so the Succ rule introduces context $v_1$, sets its core to $B_1(x)$ and $S(x, y)$, and initialises the context with (33) and (34).

We next derive (35)–(38) using hyperresolution, at which point we have sufficient information to apply hyperresolution to (28) to derive (39). Please note how the presence of (33) is crucial for this inference.

We use paramodulation to deal with equality in clause (39). As is common in resolution-based theorem proving, we order the literals in a clause and apply inferences only to maximal literals; thus, we derive (40).

Clauses (35), (36), and (40) contain function symbol $f_2$, so the Succ rule introduces context $v_2$. Due to clause (36), $B_2(x)$ holds for all ground terms that $v_2$ represents; thus, we add $B_2(x)$ to core$_{v_2}$. In contrast, atom $B_3(f_2(x))$ occurs in clause (40) in a disjunction, which means it may not hold in $v_2$; hence, we add $B_3(x)$ to the body of clause (44). The latter clause allows us to derive (45) using hyperresolution.

Clause (45) essentially says ‘$B_1(f_2(x))$ should not hold in the predecessor’, which the Pred rule propagates to $v_1$ as clause (49); one can understand this inference as hyperresolution of (40) and (45) while observing that term $f_2(x)$ in context $v_1$ is represented as variable $x$ in context $v_2$.

After two paramodulation steps, we derive clause (51), which essentially says ‘the predecessor must satisfy $B_2(x)$ or $B_3(x)$’. The set $Pr(O)$ of predecessor triggers from Definition 2 identifies this as relevant to $v_0$; the DL-clauses in (27) contain $B_2(x)$ and $B_3(x)$ in their bodies, which are represented in $v_1$ as $B_2(y)$ and $B_3(y)$. Hence $Pr(O)$ contains $B_2(y)$ and $B_3(y)$, allowing the Pred rule to derive (52).

After two more steps, we finally derive our target clause (54). We could not do this if $B_4(x)$ were maximal in (53); thus, we require all atoms in the head of a goal clause to be smallest. A similar observation applies to $Pr(O)$: if $B_3(y)$ were maximal in (50), we would not derive (51) and propagate it to $v_0$; thus, all atoms in $Pr(O)$ must be smallest too.
4 Formalising the Algorithm

In this section, we first present our consequence-based algorithm for \textsc{AlchiQ} formally, and then we present an outline of the completeness proof; full proofs are given in Bate et al. (2016).

4.1 Definitions

Our calculus manipulates context clauses, which are constructed from context terms and context literals as described in Definition 1. Unlike in general resolution, 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 in a Herbrand model, and \( y \) represents the predecessor of \( x \); this naming convention is important for the rules of our calculus. This is in contrast to the DL-clauses of an ontology, which can contain variables \( x \) and \( z_i \), where \( z_i \) refers to either the predecessor or a successor of \( x \).

Definition 1. A context \( \mathcal{F} \)-term is a term of the form \( x, y \), or \( f(x) \) for \( f \in \mathcal{F} \); a context \( \mathcal{P} \)-term is a term of the form \( B(y), B(x), f(y), S(y, x), S(x, y), S(x, f(x)) \), or \( S(\delta(x), x) \) for \( B, R \in \mathcal{P} \) and \( f \in \mathcal{F} \); and a context term is an \( \mathcal{F} \)-term or a \( \mathcal{P} \)-term. A context literal is a literal of the form \( A \approx B, B \equiv \neg y \), or \( f(x) \approx g(x) \), or \( f(x) \equiv g(y), y \equiv h, \) for \( A \) a context \( \mathcal{P} \)-term and \( \in \{=, \neq\} \). A context clause is a clause with only function-free context atoms in the body, and only context literals in the head.

Definition 2 introduces sets \( \text{Su}(O) \) and \( \text{Pr}(O) \), that identify the information that must be exchanged between adjacent contexts. Intuitively, \( \text{Su}(O) \) contains atoms that are of interest to a context’s successor, and it guides the \( \text{Succ} \) rule whereas \( \text{Pr}(O) \) contains atoms that are of interest to a context’s predecessor and it guides the \( \text{Pred} \) rule.

Definition 2. The set \( \text{Su}(O) \) of successor triggers of an ontology \( O \) is the smallest set of atoms such that, for each clause \( \Gamma \rightarrow \Delta \in O \):

- \( B(x) \in \Gamma \) implies \( B(x) \in \text{Su}(O) \),
- \( S(x, z_i) \in \Gamma \) implies \( S(x, y) \in \text{Su}(O) \), and
- \( S(z_i, x) \in \Gamma \) implies \( S(y, x) \in \text{Su}(O) \).

The set \( \text{Pr}(O) \) of predecessor triggers of \( O \) is defined as:

\[
\text{Pr}(O) = \{ A \{ x \rightarrow y, y \rightarrow x \} \mid A \in \text{Su}(O) \} \cup \{ B(y) \mid B \text{ occurs in } \text{O} \}.
\]

As in resolution, we restrict the inferences using a term order \( \triangleright \). Definition 3 specifies the conditions that the order must satisfy. Conditions 1 and 2 ensure that \( \mathcal{F} \)-terms are compared uniformly across contexts; however, \( \mathcal{P} \)-terms can be compared in different ways 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 and Nipkow 1998)—a kind of term order commonly used in equational theorem proving. Finally, condition 5 ensures that atoms that might be propagated to a context’s predecessor via the \( \text{Pred} \) rule are smallest, which is important for completeness.

Definition 3. Let \( \triangleright \) be a total, well-founded order on function symbols. A context term order \( \triangleright \) is an order on context terms satisfying the following conditions:

1. for each \( f \in \mathcal{F} \), we have \( f(x) \triangleright x \triangleright y \);
2. for all \( f, g \in \mathcal{F} \) with \( f \triangleright g \), we have \( f(x) \triangleright g(x) \);
3. for all terms \( s_1, s_2, \) and \( t \) and each position \( p \) in \( t \), if \( s_1 \triangleright s_2 \), then \( t[s_2]_p \triangleright t[s_2]_p \);
4. for each term \( s \) and each proper position \( p \) in \( s \), we have \( s \triangleright s[p]_p \);
5. for each atom \( A \equiv \psi \in \text{Pr}(O) \) and each context term \( s \notin \{ x, y \} \), we have \( A \not\triangleright s \).

Each term order is extended to a literal order, also written \( \triangleright \), as described in Section 2.

A lexicographic path order (LPO) (Baader and Nipkow 1998) over context \( \mathcal{F} \)-terms and context \( \mathcal{P} \)-terms, in which \( x \) and \( y \) are treated as constants such that \( x \triangleright y \), satisfies conditions 1 through 4. Furthermore, \( \text{Pr}(O) \) contains only atoms of the form \( B(y), S(x, y) \), and \( S(y, x) \), which we can always make smallest in the ordering; thus, condition 5 does not contradict the other conditions. Hence, an LPO that is relaxed for condition 5 satisfies Definition 3, and thus, for any given \( \triangleright \), at least one context term order exists.

Apart from orders, effective redundancy elimination techniques are critical to efficiency of resolution calculi. Definition 4 defines a notion compatible with our setting.

Definition 4. A set of clauses \( U \) contains a clause \( \Gamma \rightarrow \Delta \) up to redundancy, written \( \Gamma \rightarrow \Delta \in U \), if

1. \( \{ s \approx s', s \not\approx s' \} \subseteq \Delta \) or \( s \approx s \in \Delta \) for some terms \( s \) and \( s' \), or
2. \( \Gamma' \subseteq \Gamma \) and \( \Delta' \subseteq \Delta \) for some clause \( \Gamma' \rightarrow \Delta' \in U \).

Intuitively, if \( U \) contains \( \Gamma \rightarrow \Delta \) up to redundancy, then adding \( \Gamma \rightarrow \Delta \) to \( U \) will not modify the constraints that \( U \) represents because either \( \Gamma \rightarrow \Delta \) is a tautology or \( U \) contains a stronger clause. Note that tautologies of the form \( A \rightarrow A \) are not redundant in our setting as they are used to initialise contexts; however, whenever our calculus derives a clause \( A \rightarrow A \lor A' \), the set of clauses will have been initialised with \( A \rightarrow A \), which makes the former clause redundant by condition 2 of Definition 4. Moreover, clause heads are subjected to the usual tautology elimination rules; thus, clauses \( \gamma \rightarrow \Delta \lor s \equiv s' \) and \( \Gamma \rightarrow \Delta \lor s \equiv t \not\equiv t \) can be eliminated. Proposition 1 shows that we can remove from \( U \) each clause \( C \) that is contained in \( U \setminus \{ C \} \) up to redundancy; the \text{Elim} uses this to support clause subsumption.

Proposition 1. For \( U \) a set of clauses and \( C \) and \( C' \) clauses with \( C \in U \setminus \{ C \} \) and \( C' \in U \), we have \( C' \notin U \setminus \{ C \} \).

We are finally ready to formalise the notion of a context structure, as well as a notion of context structure soundness. The latter captures the fact that context clauses from a set \( S \), do not contain core\( _e \), in their bodies. We shall later show that our inference rules preserve context structure soundness, which essentially proves that all clauses derived by our calculus are indeed conclusions of the ontology in question.

Definition 5. A context structure for an ontology \( O \) is a tuple \( D = \{ \mathcal{V}, \mathcal{E}, \text{core}_e, \triangleright \} \), where \( \mathcal{V} \) is a finite set of contexts, \( \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{F} \) is a finite set of edges each labelled with a function symbol, function core assigns to each context \( v \) a conjunction core\( _e(v) \) of atoms over the \( \mathcal{P} \)-terms from
Su(O), function $S$ assigns to each context $v$ a finite set $S_v$ of context clauses, and function $>$ assigns to each context $v$ a context term order $>$. A context structure $D$ is sound for $O$ if the following conditions both hold.

1. For each context $v \in V$ and each clause $\Gamma \rightarrow \Delta \in S_v$, we have $O \models \text{core}_v \land \Gamma \rightarrow \Delta$.
2. For each edge $\langle u, v, f \rangle \in E$, we have $O \models \text{core}_u \rightarrow \text{core}_v \{x \mapsto f(x), y \mapsto x\}$. 

Definition 6 introduces an expansion strategy—a parameter of our calculus that determines when and how to reuse contexts in order to satisfy existential restrictions.

Definition 6. An expansion strategy is a function strategy that takes a function symbol $f$, a set of atoms $K$, and a context structure $D = (V, E, S, \text{core}, >)$. The result of strategy $(f, K, D)$ is computable in polynomial time and it is a triple $(v, \text{core}', >')$ where $\text{core}'$ is a subset of $K$; either $v \notin V$ is a fresh context, or $v \in V$ is an existing context in $D$ such that $\text{core}_v = \text{core}'$; and $>'$ is a term context order.

Simančík, Motik, and Horrocks (2014) presented two basic strategies, which we can adapt to our setting as follows.

- The eager strategy returns for each $K$ the context $v_{K_1}$ with core $K_1$. The kind of ground terms that $v_{K_1}$ represents is then very specific so the set $S_{v_{K_1}}$ is likely to be smaller, but the number of contexts can be exponential.

- The cautious strategy examines the function symbol $f$: if $f$ occurs in $O$ in exactly one atom of the form $B(f(x))$ and if $B(x) \in K_1$, then the result is the context $v_{B(x)}$ with core $B(x)$; otherwise, the result is the trivial context $v_T$ with the empty core. Context $v_{B(x)}$ is then less constrained, but the number of contexts is at most linear.

Simančík, Motik, and Horrocks (2014) discuss extensively the differences between and the relative merits of the two strategies; although their discussion deals with ALCI only, their conclusions apply to $SRIQ$ as well.

We are now ready to show soundness and completeness.

Theorem 1 (Soundness). For any expansion strategy, applying an inference rule from Table 2 to an ontology $O$ and a context structure $D$ that is sound for $O$ produces a context structure that is sound for $O$.

Theorem 2 (Completeness). Let $O$ be an ontology, and let $D = (V, E, S, \text{core}, >)$ be a context structure such that no inference rule from Table 2 is applicable to $O$ and $D$. Then, $\Gamma \rightarrow \Delta \in S_v$ holds for each query clause $\Gamma \rightarrow \Delta$ for each context $v \in V$ that satisfies conditions C1–C3.

<table>
<thead>
<tr>
<th>Table 2: Rules of the Consequence-Based Calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Core rule</strong></td>
</tr>
<tr>
<td>If $A \in \text{core}_v$, and $\top \rightarrow A \notin S_v$, then add $\top \rightarrow A$ to $S_v$.</td>
</tr>
<tr>
<td><strong>Hyper rule</strong></td>
</tr>
<tr>
<td>If $\bigwedge_{i=1}^n A_i \rightarrow \Delta \in O$, $\sigma$ is a substitution such that $\sigma(x) = x$, $\Gamma_i \rightarrow \Delta_i \cup A_i \in S_v$, $\Delta_i \cap \Gamma_i \rightarrow \Delta_i$ for $1 \leq i \leq n$, and $\bigwedge_{i=1}^n \Gamma_i \rightarrow \Delta \cup \bigvee_{i=1}^n \Delta_i \notin S_v$, then add $\bigwedge_{i=1}^n \Gamma_i \rightarrow \Delta \cup \bigvee_{i=1}^n \Delta_i \rightarrow \Delta$ to $S_v$.</td>
</tr>
<tr>
<td><strong>Eq rule</strong></td>
</tr>
<tr>
<td>If $\Gamma \rightarrow \Delta \cup \Gamma \notin S_v$, then add $\Gamma \rightarrow \Delta$ to $S_v$.</td>
</tr>
<tr>
<td><strong>Ineq rule</strong></td>
</tr>
<tr>
<td>If $\Gamma \rightarrow \Delta \cup \Gamma \notin S_v$, then add $\Gamma \rightarrow \Delta$ to $S_v$.</td>
</tr>
<tr>
<td><strong>Fact rule</strong></td>
</tr>
<tr>
<td>If $\Gamma \rightarrow \Delta \cup \Gamma \notin S_v$, then add $\Gamma \rightarrow \Delta$ to $S_v$.</td>
</tr>
<tr>
<td><strong>Elim rule</strong></td>
</tr>
<tr>
<td>If $\Gamma \rightarrow \Delta \in S_v$, then remove $\Gamma \rightarrow \Delta$ from $S_v$.</td>
</tr>
<tr>
<td><strong>Pred rule</strong></td>
</tr>
<tr>
<td>If $\langle u, v, f \rangle \in E$, $\bigwedge_{i=1}^l A_i \rightarrow \bigvee_{i=1}^{l+n} A_i \in S_v$, $\Gamma_i \rightarrow \Delta_i \cup A_i \in S_v$, $\Delta_i \cap \bigwedge_{i=1}^l A_i \rightarrow \Delta_i$ for $1 \leq i \leq l$, $A_i \in \text{Pr}(O)$ for each $l + 1 \leq i \leq l + n$, and $\bigwedge_{i=1}^l \Gamma_i \rightarrow \bigvee_{i=1}^{l+n} A_i \notin S_v$, then add $\bigwedge_{i=1}^l \Gamma_i \rightarrow \bigvee_{i=1}^{l+n} A_i \rightarrow \Delta_i \sigma$ to $S_v$, where $\sigma = {x \mapsto f(x), y \mapsto x}$.</td>
</tr>
<tr>
<td><strong>Succ rule</strong></td>
</tr>
<tr>
<td>If $\Gamma \rightarrow \Delta \cup \Gamma \in S_v$, then let $(\text{core}', &gt;') := \text{strategy}(f, K_1, D)$; if $v \in V$, then let $\rightarrow_v := \rightarrow_v \cap &gt;'$, and otherwise let $\rightarrow_v := \bigvee {v}$,$\rightarrow_v := \rightarrow', \text{core}_v := \text{core}',$ and $S_v := \emptyset$; add the edge $(u, v, f)$ to $E$; and add $A' \rightarrow A'$ to $S_v$, for each $A' \in K_2 \setminus \text{core}_v$, where $\sigma = {x \mapsto f(x), y \mapsto x}$, $K_1 = {A' \in \text{Su}(O) \mid \Gamma \rightarrow A' \in S_v}$, and $K_2 = {A' \in \text{Su}(O) \mid \Gamma' \rightarrow \Delta' \cup A' \in S_v}$ and $\Delta' \notin S_v$.</td>
</tr>
</tbody>
</table>
A2. Introduce a context \( q \) into \( D \); set \( \text{core}_q = \Gamma_Q \); for each \( A \in \Gamma_q \), add \( \top \rightarrow A \) to \( S_q \) to satisfy condition C3; and initialise \( \succ_q \) in a way that satisfies condition C2.

A3. Apply the inference rules from Table 2 to \( D \) and \( O \).

A4. \( \Gamma_Q \rightarrow \Delta_Q \) holds if and only if \( \Gamma_Q \rightarrow \Delta_Q \in S_v \).

Propositions 2 and 3 show that our calculus is worst-case optimal for both \( \mathcal{ALCHIQ} \) and \( \mathcal{ELH} \).

**Proposition 2.** For each expansion strategy that introduces at most exponentially many contexts, algorithm A1–A4 runs in worst-case exponential time.

**Proposition 3.** For \( \mathcal{ELH} \) ontologies and queries of the form \( B_1(x) \rightarrow B_2(x) \), algorithm A1–A4 runs in polynomial time with either the cautious or the eager strategy; and with the cautious strategy and the Hyper rule applied eagerly, the inferences in step A3 correspond directly to the inferences of the \( \mathcal{ELH} \) calculus by Bauder, Brandt, and Latz (2005).

4.2 An Outline of the Completeness Proof

To prove Theorem 2, we fix an ontology \( O \), a context structure \( D \), a query clause \( \Gamma_Q \rightarrow \Delta_Q \), and a context \( q \) such that properties C2 and C3 of Theorem 2 are satisfied and \( \Gamma_Q \rightarrow \Delta_Q \notin S_v \); we then construct a Herbrand interpretation that satisfies \( O \) but refutes \( \Gamma_Q \rightarrow \Delta_Q \). We reuse techniques from equational theorem proving (Nieuwenhuis and Rubio 1995) and represent this interpretation by a rewrite system \( R \) — a finite set of rules of the form \( l \Rightarrow r \). Intuitively, such a rule says that that any two terms of the form \( f_1(\ldots f_n(l)) \ldots \) and \( f_1(\ldots f_n(r)) \ldots \) with \( n \geq 0 \) are equal, and that we can prove this equality in one step by rewriting (i.e., replacing) \( l \) with \( r \). Rewrite system \( R \) induces a Herbrand equality interpretation \( R^* \) that contains each \( t \approx r \) for which the equality between \( l \) and \( r \) can be verified using a finite number of such rewrite steps. The universe of \( R^* \) consists of \( F \)- and \( P \)-terms constructed using the symbols in \( F \) and \( P \), and a special constant \( c \); for convenience, let \( T \) be the set of all \( F \)-terms from this universe.

We obtain \( R \) by unfolding the context structure \( D \) starting from context \( q \): we map each \( F \)-term \( t \in T \) to a context \( \Gamma_t \) in \( D \), and we use the clauses in \( S_{\Gamma_t} \) to construct a model fragment \( R_t \) — the part of \( R \) that satisfies the DL-clauses of \( O \) when \( x \) is mapped to \( t \). The key issue is to ensure compatibility between adjacent model fragments: when moving from a predecessor term \( t' \) to a successor term \( t = f(t') \), we must ensure that adding \( R_{t'} \) to \( R_t \) does not affect the truth of the DL-clauses of \( O \) 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 \( \Gamma_{t'} \rightarrow \Delta_{t'} \); conjunction \( \Gamma_{t'} \) contains atoms that are ‘inherited’ from \( t' \) and so must hold at \( t' \); and disjunction \( \Delta_{t'} \) contains atoms that must not hold at \( t' \) because \( t' \) relies on their absence.

The model fragment construction takes as parameters a term \( t \), a context \( v = X_t \), and a clause \( \Gamma_t \rightarrow \Delta_t \). Let \( N_t \) be the set of ground clauses obtained from \( S_{\Gamma_t} \) by mapping \( x \) to \( t \) and \( y \) to the predecessor of \( t \) (if it exists), and whose body is contained in \( \Gamma_t \). Moreover, let \( S_{\Gamma_t} + \text{Pr}(O) \) be obtained from \( S_{\Gamma_t} + \text{Pr}(O) \) by mapping \( x \) to \( t \) and \( y \) to the predecessor of \( t \) if one exists; thus, \( S_{\Gamma_t} \) contains the ground atoms of interest to the successors of \( t \), and \( \text{Pr}(O) \) contains the ground atoms of \( \text{interest to the predecessor of } t \). The model fragment for \( t \) can be constructed if properties L1–L3 hold:

L1. \( \Gamma_t \rightarrow \Delta_t \notin N_t \).

L2. If \( t = c \), then \( \Delta_t = \Delta_Q \); and if \( t \neq c \), then \( \Delta_t \subseteq \text{Pr}(O) \).

L3. For each \( A \in \Gamma_t \), we have \( \Gamma_t 
 A \notin N_t \).

The construction produces a rewrite system \( R_t \) such that:

F1. \( R_t^* \models N_t \), and

F2. \( R_t^* \not\models \Gamma_t \rightarrow \Delta_t \)—that is, all of \( \Gamma_t \), but none of \( \Delta_t \), hold in \( R_t^* \), and so the model fragment at \( t \) is compatible with the ‘inherited’ constraints.

We construct rewrite system \( R_t \) by adapting the techniques from paramodulation-based theorem proving. First, we order all clauses in \( N_t \) into a sequence \( C^i = \Gamma^i \rightarrow \Delta^i \lor L^i \), \( 1 \leq i \leq n \), that is compatible with the context ordering \( \succ \) in a particular way. Next, we initialise \( R_t \) to \( \emptyset \), and then we examine each clause \( C^j \) in this sequence: if \( C^j \) does not hold in the model constructed thus far, we make the clause true by adding \( L^j \) to \( R_t \). To prove condition F1, we assume for the sake of a contradiction that a clause \( C^j \) with smallest \( i \) exists such that \( R_t^* \not\models C^j \), and we show that an application of the Eq, Ineq, or Factor rule to \( C^j \) necessarily produces a clause \( C^k \) such that \( R_t^* \not\models C^k \) and \( j < k \). 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 \( \Delta_t \) 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 \( \Gamma_t \) are actually produced in \( R_t^* \).

To obtain \( R_t \), we inductively unfold \( D \), and at each step we apply the model fragment construction to the appropriate parameters. 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 property C3 of Theorem 2. For the induction step, we assume that we have already mapped some term \( t' \) to a context \( u = X_{t'} \), and we consider term \( t = f(t') \) for each \( f \in F \).

- 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 \( D \) contains an edge \( \langle u, v, f \rangle \) such that \( v \) satisfies all preconditions of the rule, so we define \( X_v = v \). Moreover, we let \( \Gamma_t = R_{t'} \cap \text{Su}(O) \) be the set of atoms that hold at \( t' \) and are relevant to \( t \), and we let \( \Delta_t = \text{Pr}(O) \setminus R_{t'} \) be the set of atoms that do not hold at \( t' \) and are relevant to \( t \). We finally show that such \( \Gamma_t \) and \( \Delta_t \) satisfy condition L1: otherwise, the Pred rule derives a clause in \( N_{t'} \) that is not true in \( R_{t'}^* \).

After processing all relevant terms, we let \( R \) be the union of all \( R_t \) from the above construction. To show that \( R^* \) satisfies \( O \), we consider a DL-clause \( \Gamma \rightarrow \Delta \in O \) and a substitution \( \tau \) that makes the clause ground. W.l.o.g. we can
We have implemented our calculus in a prototype system called Sequoia. The calculus was implemented exactly as presented in this paper, with no optimisation other than a suitable indexing scheme for clauses. The system is written in Scala, and it can be used via the command line or the OWL API. It currently handles the $\text{SRTQ}$ subset of OWL 2 DL (i.e., it does not support datatypes, nominals, or reflexive roles), for which it supports ontology classification and concept satisfiability; other standard services such as ABox realisation are currently not supported.

We have evaluated our system using the methodology by Steigmiller, Liebig, and Glimm (2014) by comparing Sequoia with HermiT 1.3.8, Pellet 2.3.1, FaCT++ 1.6.4, and Konclude 1.6.1. We used all reasoners in single-threaded mode in order to compare the underlying calculi; moreover, Sequoia was configured to use the cautious strategy. All systems, ontologies, and test results are available online.\footnote{http://krr-nas.cs.ox.ac.uk/2015/KR/cr/}

We used the Oxford Ontology Repository\footnote{http://www.cs.ox.ac.uk/issg/ontologies/} from which we excluded 7 ontologies with irregular RBoxes. Since Sequoia does not support datatypes or nominals, we have systematically replaced datatypes and nominals with fresh classes and data properties with object properties, and we have removed ABox assertions. We thus obtained a corpus of 777 ontologies on which we tested all reasoners.

We run our experiments on a Dell workstation with two Intel Xeon E5-2643 v3 3.4 GHz processors with 6 cores per processor and 128 GB of RAM running Windows Server 2012 R2. We used Java 8 update 66 with 15 GB of heap memory allocated to each Java reasoner, and a maximum private working set size of 15 GB for each reasoner in native code. In each test, we measured the wall-clock classification time; this excludes parsing time for reasoners based on the OWL API (i.e., HermiT, Pellet, FaCT++, and Sequoia). Each test was given a timeout of 5 minutes. We report the average time over three runs, unless an exception or timeout occurred in one of the three runs, in which case we report failure.

Figure 3 shows an overview of the classification times for the entire corpus. The $y$-axis shows the classification times in logarithmic scale, and timeouts are shown as infinity. A number $n$ on the $x$-axis represents the $n$-th easiest ontology for a reasoner with ontologies sorted (for that reasoner) in the ascending order of classification time. For example, a point (50, 100) on a reasoner’s curve means that the 50th easiest ontology for that reasoner took 100 ms to classify.

Sequoia could process most ontologies (733 out of 784) in under 10s, which is consistent with the other reasoners. The system was fairly robust, failing on only 22 ontologies; in contrast, HermiT failed on 42, Pellet on 138, FaCT++ on 132, and Konclude on 8 ontologies. Moreover, Sequoia succeeded on 21 ontologies on which all of HermiT, Pellet and FaCT++ failed. Finally, there was one ontology where Sequoia succeeded and all other reasoners failed; this was a hard version of FMA (ID 00285) that uses both disjunctions and number restrictions.

Figure 4 shows an overview of how each reasoner performed on each type of ontology. We partitioned the ontologies in the following four groups: within a profile of OWL 2 DL (i.e., captured by OWL 2 EL, QL, or RL); Horn but not in a profile; disjunctive but without number restrictions; and disjunctive with number restrictions. We used the OWL API to determine profile membership, and we identified the remaining three groups after structural transformation. In addition, for each reasoner, we categorise each ontology as either ‘easy’ (< 10s), ‘medium’ (10s to 5min), and ‘hard’ (timeout or exception). The figure depicts a bar for each reasoner and group, where each bar is divided into blocks representing the percentage of ontologies in each of the aforementioned categories of difficulty. For Sequoia, over 98% of
profile ontologies and over 91% of out-of-profile Horn ontologies are easy, with the remainder being of medium difficulty. Sequoia timed out largely on ontologies containing both disjunctions and equality, and even in this case only Konclude timed out in fewer cases.

In summary, although only an early prototype, Sequoia is a competitive reasoner that comfortably outperforms HermiT, Pellet, and FaCT++, and which exhibits a nice pay-as-you-go behaviour. Furthermore, problematic ontologies seem to mostly contain complex RBoxes or large numbers in cardinality restrictions, which suggests promising directions for future optimisation.

6 Conclusion and Future Work
We have presented the first consequence based calculus for $SRTQ$—a DL that includes both disjunction and counting quantifiers. Our calculus combines ideas from state of the art resolution (and (hyper)tableau calculi, including the use of ordered paramodulation for equality reasoning. Despite its increased complexity, the calculus mimics existing calculi on $EL$ ontologies. Although it is an early prototype with plenty of room for optimisation, our system Sequoia is competitive with well-established reasoners and it exhibits nice pay-as-you-go behaviour in practice.

For future work, we are confident that we can extend the calculus to support role reflexivity and datatypes, thus handling all of OWL 2 DL except nominals. In contrast, handling nominals seems to be much more involved. In fact, adding nominals to $ACIHIQ$ raises the complexity of reasoning to NEXPTIME so a worst-case optimal calculus must be nondeterministic, which is quite different from all consequence-based calculi we are aware of. Moreover, a further challenge is to modify the calculus so that it can effectively deal with large numbers in number restrictions.

References