A Consequence-based Algebraic Calculus for $\mathcal{SHOQ}$

Nikoo Zolfaghar Karahroodi and Volker Haarslev

Concordia University, Montréal, Canada
n_zolfa@encs.concordia.ca, haarslev@cse.concordia.ca

Abstract. In this paper, we present a novel consequence-based algorithm to perform subsumption reasoning in $\mathcal{SHOQ}$, which support nominals and Qualified Cardinality Restrictions (QCRs). Our algorithm maps numerical restrictions imposed by QCRs or nominals to inequalities and determines the feasibility of inequality systems by means of Integer Linear Programming.

1 Introduction and Motivation

Most modern DL reasoners, such as HermiT [8], FaCT++ [14], Pellet [13], and RacerPro [5], implement highly optimized *tableau-based algorithms*, or its variations. The main idea of tableau-based algorithms for classification is to systematically construct a model of the input ontology plus the negation of each subsumption candidate. If all constructed models by the procedure turn out to contain an *obvious contradiction* (clash), one can conclude that the subsumption candidate holds.

There is another type of reasoning algorithms, called *consequence based algorithms*, which instead of building counter-models (like tableau-based algorithms), directly derive logical consequences of axioms in the ontology using inference rules. These algorithms never check for subsumptions that are not entailed. Typically, the number of entailed subsumptions is much smaller than the number of potential subsumptions between ontology concepts. These algorithms can classify an ontology in a single pass.

Consequence-based (CB) algorithms were first introduced for the DL $\mathcal{EL}$ [1]. These algorithms later were extended to DL $\mathcal{ELO}$ [7] and Horn-$\mathcal{SHIQ}$ [6] – DLs that support *nominals* and functional roles respectively, but not disjunctive reasoning. The framework for consequence based calculi was proposed for $\mathcal{ALCI}$ [12], which support disjunctive reasoning, but not *Qualified Cardinality Restrictions* (QCRs). Recently this framework was extended to DL $\mathcal{SRIQ}$ that supports QCRs [2], but their time complexity is dependent to the number of QCRs and the values occurring in them.

However, to the best of our knowledge, none of the existing CB algorithms could handle the combination of QCRs and nominals efficiently. As we discuss in Section 3, it is challenging to extend these algorithms to DLs such as $\mathcal{SHOQ}$ that combines different kinds of numerical restrictions: QCRs and nominals.

On the other hand, most existing DL reasoners try to satisfy imposed numerical constraints by exhausting all possibilities. Merely searching for a model in such an arithmetically uninformed or blind way is usually very inefficient. Using arithmetic methods can improve the average case performance of numeric reasoning. Employing this technique for tableau-based reasoning in $\mathcal{SHOQ}$ has shown to have an impressive practical result [4]. The main drawback of this approach is producing an exponential number of variables.

In Section 3 we extend the framework introduced in [12] to develop a CB calculus for $\mathcal{SHOQ}$. Our algorithm employs *Integer Linear Programming* to properly handle numerical features of the language, while eliminating its drawbacks in generating numerous variables, by applying optimization techniques, specifically *column and row generation*. Thus, although a practical evaluation of our calculus is still pending, we believe that it is most likely perform well in practice on $\mathcal{SHOQ}$ ontologies.

2 Preliminaries

Description Logics $\mathcal{ALCHOQ}$ and $\mathcal{SHOQ}$ are defined w.r.t. non-empty and disjoint sets of *atomic concepts* N_C, *roles* N_R and *nominals* N_o.

Ontologies are interpreted using Tarski-style semantics. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set of elements called the domain, and $.^{\mathcal{I}}$ is an interpretation function that maps each atomic concept $A \in N_C$ to a subset of $\Delta^{\mathcal{I}}$, and each role $R \in N_R$ to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

A Qualified Cardinality Restriction (QCR), also called qualified number restriction, specifies a lower ($\geq nR.C$) or upper ($\leq nR.C$) bound on the number of *R-successors* belonging to a certain concept C, where $R \in N_R$. A domain element $y \in \Delta^{\mathcal{I}}$ is said to be an R-successor if there exists a domain element $x \in \Delta^{\mathcal{I}}$ such that x and y are related trough the role R, $\langle x, y\rangle \in R^{\mathcal{I}}$. The set of all R-successors for a given role R is defined as $Succ(R) = \{y \in \Delta^{\mathcal{I}} \mid \langle x, y\rangle \in R^{\mathcal{I}}\}$. A qualifying concept A is a concept name that occurs in a QCR of the form $\leq nR.A$ or $\geq nR.A$ to impose a minimum or maximum on the number of R-successors for a role $R \in N_R$.

Nominals are known as named individuals. They are also considered as concepts with exactly one instance that will be interpreted as singleton sets. Nominals enable DLs with the notion of uniqueness and identity. There exist many concepts in the real world that need to be modeled using *nominals* such as "Moon", "Earth" or "Canada".

A literal L is a concept of the form C, o, $\exists R.C$, $\forall R.C$, $\geq nR.C$, $\leq nR.C$, for $C \in N_C$, $o \in N_o$ and $R \in N_R$. The set of all literals is denoted as N_L. Through the rest of this paper we denote a conjunction and disjunction of literals with K and M, respectively. Furthermore, we identify them as a set of literals and use them in standard set operations. The conjunction (disjunction) of literals may be empty, which is abbreviated as $\top$ ($\bot$). An *axiom* is an expression of the form $C_1 \sqsubseteq C_2$ (*general concept inclusion* (GCI)), $R_1 \sqsubseteq R_2$ (*role inclusion*),

or $Trans(R)$ (*role transitivity axiom*). A $\mathcal{SHOQ}$ ontology $\mathcal{O}$ is a finite set of axioms.

A *clause* is a general concept inclusion of the form $\bigsqcap_{i=1}^{m} L_i \sqsubseteq \bigsqcup_{i=m+1}^{n} L_i$ where $0 \leq m \leq n$ and each L_i is a literal. A clause is *normal* if each L_i with $1 \leq i \leq m$ is an atomic concept and a clause is a *query* if each L_i with $m+1 \leq i \leq n$ is an atomic concept. In a clause of the form $K \sqsubseteq M$, the conjunction K is called the *antecedent*, and the disjunction M is called the *consequence*. An ontology $\mathcal{O}$ is *normalized* if each GCI in $\mathcal{O}$ is a normal clause. The ontology $\mathcal{O}$ can be converted to a normalized ontology $\mathcal{O}'$ in linear time such that $\mathcal{O}'$ is a conservative extension of $\mathcal{O}$. This conversion is known as *structural transformation* (see, e.g., [6,8]), so we eliminate the details due to space restrictions. A normalized $\mathcal{SHOQ}$ ontology can be rewritten to a normal $\mathcal{ALCHOQ}$ ontology by eliminating all role transitivity axioms [10]. Given a set of literals $\boldsymbol{L}$, a clause $K \sqsubseteq M$ is over $\boldsymbol{L}$ if $K \cup M \subseteq \boldsymbol{L}$. A clause $K' \sqsubseteq M'$ is a strengthening of a clause $K \sqsubseteq M$ if $K' \subseteq K$ and $M' \subset M$. We use $K \sqsubseteq M \in_* \mathcal{C}$ to show that a set of clauses $\mathcal{C}$ contains at least one strengthening of $K \sqsubseteq M$.

Due to the presence of nominals, assertions can be transformed to terminological knowledge (TBox axioms). So we only consider terminological reasoning in this research. In $\mathcal{SHOQ}$ the interaction of transitive roles with number restrictions would cause undecidability. To avoid this interaction, $\mathcal{SHOQ}$ allows number restrictions only with *simple roles* which are neither transitive nor have transitive sub-roles.

3 A consequence-based calculus for $\mathcal{SHOQ}$

In this section, we explain our consequence-based algorithm for classifying a normalized $\mathcal{ALCHOQ}$ ontology $\mathcal{O}$ having a finite set of queries $\mathcal{Q}$ to determine whether $\mathcal{O} \models q$ holds for each query $q = K \sqsubseteq M \in \mathcal{Q}$. Since the goal of the algorithm is to obtain the *ontology classification*, we only consider queries of the form $C_1 \sqsubseteq C_2$ where C_1 and C_2 are atomic concepts from $\mathcal{O}$. A formal description of our algorithm is presented in Section 3.2.

3.1 Intuition

This section demonstrate our algorithm at a high level by applying it to the ontology $\mathcal{O}$ and the query q as specified in Example 1 to prove that $\mathcal{O} \models q$.

Example 1. Let $\mathcal{O}$ be an ontology containing axioms (1) – (9), and $q = A \sqsubseteq D$. One can readily verify that $\mathcal{O} \models q$ due to cardinality restrictions imposed on the concepts A, C by merging nominals o_1, o_2 and o_3.

$A \sqsubseteq \exists R.o_1$ (1)	$A \sqsubseteq \exists R.o_3 \sqcup D$ (3)	$o_1 \sqsubseteq C$ (5)	$o_1 \sqsubseteq \geq 2S.B$ (8)
$A \sqsubseteq \exists R.o_2$ (2)	$A \sqsubseteq \leq 1R.C$ (4)	$o_2 \sqsubseteq C$ (6)	$B \sqsubseteq o_1 \sqcup o_2 \sqcup o_3$ (9)
		$o_3 \sqsubseteq C$ (7)	

Our consequence-based algorithm constructs a graph whose vertices are called *nodes*. Each node describes a set of elements in the model $\mathcal{I}$, which is an arbitrary model. The edges between nodes represent *role successor* relations between the corresponding elements. Each node v is associated with an atomic concept $\texttt{core}(v)$. Intuitively, $\texttt{core}(v)$ holds for every element of $\mathcal{I}$ that corresponds to v. Moreover, each node v is labeled with a set of clauses $\mathcal{L}(v)$. Each clause $K \sqsubseteq M \in \mathcal{L}(v)$ is related to $\texttt{core}(v)$ and should be interpreted as $\texttt{core}(v) \sqcap K \sqsubseteq M$.

Figure 1 illustrates the constructed graph for Example 1, where each node is shown as a circle. The core concept of each node v is shown as the subscript of v inside the circle, and the label of each node, $\mathcal{L}(v)$, is shown either below or above the circle. The numbers next to the clauses in $\mathcal{L}(v)$ correspond to the order of inference rule applications.

The inference process starts by initializing the algorithm according to the target query. Since in this example the query is $q = A \sqsubseteq D$, we introduce a node v_A where $\texttt{core}(v_A) = A$, and add clause (10) to $\mathcal{L}(v_A)$. Intuitively, this clause states that there has to be at least one element in the model $\mathcal{I}$ that satisfies A. Since nominals always exist, we also need to create one node v_o, for representing each nominal o occurring in $\mathcal{O}$. Therefore nodes v_{o_i} for $1 \leq i \leq 3$ are created to represent existing nominals and are initialized by clauses (11) – (13).

Afterwards, the `subs` rule (see Table 1) derives clauses (14) – (20). Since only A is assumed to hold in v_A, the algorithm derives only a linear number of clauses. This rule is analogous to the $R_{\sqsubseteq}$ rule in the original completion rules for $\mathcal{EL}$ [1] and is adapted from [12].

Clause (14) and (15) state that v_A must have R-successors ${o_1}^{\mathcal{I}}$ and ${o_2}^{\mathcal{I}}$ respectively. Clause (16) says that all the corresponding elements to v_A must have an R-successor ${o_3}^{\mathcal{I}}$ or should satisfy D. On the other hand, clause (17) indicates that all the corresponding elements to v_A must have at most one R-successor in which C holds.

The `Nom` rule finds *a solution* ξ that satisfies all these restrictions by means of an *Integer Linear Programming (ILP) Component*. This component is called whenever a QCR is added to the label of a node. Before feeding numerical restrictions to the ILP component, every number restriction should be encoded into an inequality. The encoding procedure will be discussed thoroughly in Section 4. The ILP component may also receive other information such as the subsumption relations in v_{o_1} and v_{o_2} in order to find a more constrained solution that is less likely to cause a contradiction later due to added entailments. The atomic concepts that participate in a disjunction with QCRs would not be reflected in inequalities, e.g., concept D in clause (16).

If the inequality system is feasible, the ILP component returns a possible solution which satisfies all numerical restrictions. The returned solution is a set of tuples each containing a partition element and its cardinality (see Section 4). In Example 1, the ILP component returns $\xi(v_A) = \{\langle\{o_1, o_2, C\}, 1\rangle\}$. Which means that, for satisfying the submitted constraints (14)-(17), there should exist one element in the domain in which o_1, o_2 and C hold. Due to the presence of

nominals in the returned solution the `Nom` rule is applicable, which essentially checks if a nominal o always would appear in a partition element p together with a concept $C \in p$ (or another nominal $o_2 \in p$). Accordingly, the `Nom` rule is applicable for nominals o_1 and o_2 and derives clauses (21) and (22).

The `Nom` rule does not apply to nominal o_3, because clause (16) represents a possibility and if D is true in (16) then o_3 does not have to exist in a partition element with o_1 and o_2. However, the consequences of assuming o_3 to be equal to o_1 and o_2 still have to be checked. To reflect this possible equality in the graph, the `Sigma` rule adds clauses (26) – (29) to the graph. In general, the `select` function chooses a concept X of each partition element p as its representative. The concept X would be the core of a node v ($\texttt{core}(v) = X$). A clause $D \sqsubseteq D$ is added to the label of a node v_X for every other concept D that holds in p, which reflects the possibility of holding D for elements corresponding to v_X.

Clauses (26) and (21) state that o_1, o_2 and o_3 are possibly merged, so, based on Clause (9) the cardinality of B would to be less than or equal to one. Clause (30) requires at least two elements in $B^{\mathcal{I}}$. One can verify that these constraints are infeasible altogether. The ILP component discovers this infeasibility and returns a set of *clash culprits* (Definition 7) that cause the contradiction. A clash culprit is the *minimum set* of literals such that their conjunction causes an infeasibility.

In this example, clauses $o_1 \sqsubseteq o_2$, $o_1 \sqsubseteq o_3$, $o_1 \sqsubseteq \geq 2R.B$ and $B \sqsubseteq o_1 \sqcup o_2 \sqcup o_3$ would be returned as a *clash culprit* altogether. But $o_1 \sqsubseteq o_3$ is the only *possible* clause (shown by clause (26) and (28)) that participates in the contradiction. One can see that all (i.e. in our example, the only one) possible clauses that cause the clash exist in the label of the same node ($\mathcal{L}(o_1)$). Thus applying the `Bottom` rule adds clause (31) – (34) to the graph.

Clause (31) states that nominals o_1, o_3 can not hold (be merged) together, which makes the inequality system containing clauses (14) - (17) infeasible. There is only one possible clause in the returned clash culprits which is $A \sqsubseteq \exists R.o_1$ and it belongs to $\mathcal{L}(v_A)$, so, the `Bottom` rule produces clause (35).

Later, the inequality system corresponding to clauses (30) and (21) becomes feasible. Based on the solution returned by ILP, the `Sigma` rule requires two nodes corresponding to o_2 and o_3. But then, there is no need to introduce fresh nodes with cores o_2 and o_3 because our algorithm is designed to reuse the existing nodes v_{o_2} and v_{o_3}. Consequently, we introduce edges (36) and (37) using the `Sigma` rule.

Our *reuse* strategy is comparable to blocking techniques in tableau-based algorithms but is much more efficient in eliminating redundant computations. First, our algorithm never needs more than a linear number of nodes to the number of concepts occurring in ontology $\mathcal{O}$, whereas tableau-based algorithms can construct trees of double exponential size. Second, the clauses in the label of each node are not localized to a specific element in $\mathcal{I}$, and so our algorithm draws the inferences for a particular core only once.

Eventually, we use the `Sigma` rule to add clause (38), (39), and (40). At this point no further inference rule is applicable. Since all clauses are "relative" to the

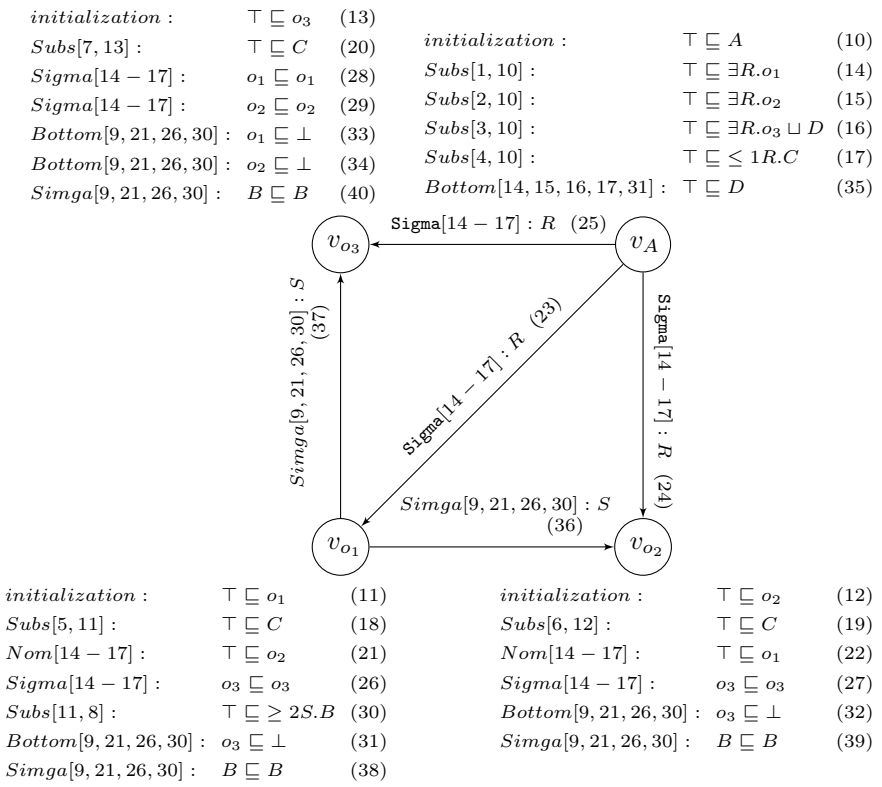

Fig. 1. Example inferences of the consequence-based algorithm

core of the corresponding node, clause (35) corresponds to $A \sqsubseteq D$, so we have proved $\mathcal{O} \models A \sqsubseteq D$. In fact, due to (35), we know that $\mathcal{O} \models K \sqsubseteq M$ for each query $K \sqsubseteq M$ such that $A \sqsubseteq D$ is a strengthening of $K \sqsubseteq M$.

We are aware that an unrestricted application of the `Subs` rule can overwhelm the whole graph with produced clauses. So, our next step will be to extend our algorithm to employ an ordered resolution variant addressing this problem [9].

3.2 Formalization

In the following, we provide a formal description of our consequence-based reasoning algorithm by introducing a set of inference rules shown in Table 1, the `Subs` rule is adapted from [12] but the rest of rules are original. Before formally defining our algorithm, we present some notions which are used later in formalizing our inference rules.

Definition 1 (Completion Graph). *A* Completion Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ *is composed of a set of* nodes. *Each node v is labeled by an atomic concept C*

which is called $\texttt{core}(v)$ *and a set of clauses* $\mathcal{L}(v)$*. Each edge* $\langle u, v\rangle \in \mathcal{E}$ *is labeled by the set* $\mathcal{L}(\langle u, v\rangle) \subseteq \mathcal{R}$*.*

In addition, $\mathcal{G}$ is over a set of literals $\boldsymbol{L} \subseteq N_L$ if $\boldsymbol{L}$ contains all literals in $\mathcal{G}$. Through the rest of this paper we fix $\boldsymbol{L}$ to contain the set of literals occurring in $\mathcal{O} \cup \mathcal{Q}$ since $\mathcal{O}$ and $\mathcal{Q}$ are finite, set $\boldsymbol{L}$ is finite as well. Assuming v to be an arbitrary node in $\mathcal{G}$ and $q = K \sqsubseteq M$ to be an arbitrary query; then v *covers* q if $\texttt{core}(v) \subseteq K$.

Definition 2 (Clause System). *A clause system for a completion graph* $\mathcal{G}$ *is a function* $\mathcal{L}$ *that assigns a set of clauses* $\mathcal{L}(v)$ *to each node* $v \in \mathcal{V}$*. It is possible to decide query entailment based on* $\mathcal{L}$*: A query* $q = K \sqsubseteq M$ *is entailed,* $\mathcal{O} \models K \sqsubseteq M$*, if and only if* $K \sqsubseteq M \in \mathcal{L}(v)$*, for each node* v *covering* q *and initialized so that* $K \sqsubseteq L \in_* \mathcal{L}(v)$ *holds for each literal* $L \in K$*.*

Theorem 1 states that all clauses derived by our calculus are indeed conclusions of the input ontology. The completeness of our algorithm is ensured by Theorem 2.

Definition 3 (Soundness). *A completion graph* $\mathcal{G}$ *is sound for* $\mathcal{O}$*, if for every edge* $R \in \mathcal{L}(\langle v, u\rangle)$*, there exist arbitrary elements* $\alpha, \beta \in \Delta$ *such that* $\alpha \in (\texttt{core}(v))^{\mathcal{I}}$*,* $\beta \in (\texttt{core}(u))^{\mathcal{I}}$ *and* $\mathcal{O} \models \langle \alpha, \beta\rangle \in R^{\mathcal{I}}$*. A clause system* $\mathcal{L}$ *is sound for* $\mathcal{O}$ *if* $\mathcal{O} \models \texttt{core}(v) \sqcap K \sqsubseteq M$ *holds for each node* $v \in \mathcal{V}$ *and each clause* $K \sqsubseteq M \in \mathcal{L}(v)$*.*

Theorem 1 (Soundness). *Let* $\mathcal{G}_1$ *be a completion graph and let* $\mathcal{L}_1$ *be a clause system for* $\mathcal{G}_1$*. The completion graph* $\mathcal{G}_2$ *and the clause system* $\mathcal{L}_2$ *are obtained by applying an inference rule from Table 1 to* $\mathcal{G}_1$ *and* $\mathcal{L}_1$*. If* $\mathcal{G}_1$ *and* $\mathcal{L}_1$ *are sound for* $\mathcal{O}$*, then both* $\mathcal{G}_2$ *and* $\mathcal{L}_2$ *are sound for* $\mathcal{O}$*.*

Theorem 2 (Completeness). *Let* $\mathcal{G}$ *be a completion graph and let* $\mathcal{L}$ *be a clause system for* $\mathcal{G}$ *such that no inference rule from Table 1 is applicable to* $\mathcal{G}$ *and* $\mathcal{L}$*. Then* $K \sqsubseteq M \in_* \mathcal{L}(v)$ *hold for each query* $q = K \sqsubseteq M$ *and each node* $v \in \mathcal{V}$ *if* $\mathcal{O} \models K \sqsubseteq M$ *and* $K \sqsubseteq L \in_* \mathcal{L}(v)$ *for each literal* $L \in K$*.*

Definition 4 (Known/Possible QCRs). *The set* $\boldsymbol{Q}_k(v)$ *contains QCRs which are known to hold in a node* v *for which* $\top \sqsubseteq \leq nR.C$ *or* $\top \sqsubseteq \geq nR.C$ *has been derived in node* v*. Set* $\boldsymbol{Q}_p(v)$ *contains QCRs that can possibly hold in a node* v*. These are those QCRs for which* $K \sqsubseteq \leq nR.L \sqcup M$ *or* $K \sqsubseteq \geq nR.L \sqcup M$ *has been derived in node* v *for some* $K \not\equiv \top$ *or* $M \not\equiv \bot$*. The set* $\boldsymbol{Q}_p(v)$ *is formally defined as* $\boldsymbol{Q}_p(v) = \{\bowtie nR.C \mid K \sqsubseteq \bowtie nR.C \sqcup M \in \mathcal{L}(v), K \not\equiv \top \text{ or } M \not\equiv \bot\}$*.*

Definition 5 (Possible Clause). *The set* $\boldsymbol{C}_p(v)$ *contains* possible *clauses that are considered to hold in a node* v*. A clause* $K \sqsubseteq M \in \mathcal{L}(v)$ *is a* possible *clause, if either* $K \equiv \top$ *or* $M \equiv \bot$*, otherwise, it would be a* known *clause. The set of known clauses in the label of a node* v *is denoted as* $\boldsymbol{C}_k(v)$*.*

Definition 6 (`select` function). *The* `select` *function chooses one qualifying concept $C \in p$ as the core, where C is either a qualifying concept in a* known at-least restriction *or a nominal, and p a partition element. If p contains a nominal o, then the* `select` *function always selects the nominal o as the representative for p. But if p contains more that one nominal then all of them are selected, one after the other. The consequences of the rule are applied to all these nominals. The method* `select` *is called a function because it returns a unique representative for all partition elements containing the same set of concepts.*

Definition 7 (Clash Culprits ($CC(v)$)). *A* Clash Culprit *set ($CC(v) = \{CC_1(v), ..., CC_n(v)\}$) is returned by the ILP component if an inequality system is infeasible. A $CC_i, 1 \leq i \leq n$ consists of the minimum number of clauses such that the inequality system containing the corresponding inequalities is infeasible.*

Note that there may be more than one clash culprit for the same set of inequalities. For example assume we are trying to check the satisfiability of the QCRs: $\geq 2R.C, \geq 3R.C$ and $\leq 1R.C$. One can recognize two clash culprits in $CC(v)$ that are involved in the clash, $CC_1 = \{\geq 2R.C, \leq 1R.C\}$ and $CC_2 = \{\geq 3R.C, \leq 1R.C\}$. Considering each set of clash culprits may lead to a subsumption. Accordingly, we need to apply the `Bottom` rule for each CC_i.

If all the clauses in a clash culprit are known information, we call it a **strict** unsatisfiablity, which can not be avoided by choosing an other possibility. The `Bottom` rule only considers possible clauses in a clash culprit set because known clauses can never be violated. Therefore, if all the clash culprits are known then it means that there is an unsatisfiability in the ontology. All clauses of a clash culprit set may not necessarily occur in the label of one particular node. In this case, no clauses can be derived based on the current knowledge.

4 Integer Linear Programming Component

We implement the ILP services in a separate module called the *ILP component*. The task of the ILP Component can be divided into three parts: producing variables, encoding numerical restrictions into inequalities and checking the satisfiability of derived inequalities.

The ILP component would be called for checking the satisfiablity of QCRs in the label of a node v. Each time, all QCR occurring in the label of v and all clauses in $\boldsymbol{C}_k(v_C)$, $\boldsymbol{C}_k(v_o)$ and $\boldsymbol{C}_p(v_o)$ are fed to ILP component, where $C \in Q_v$ and $o \in N_o(v)$.

4.1 Decomposition Set

A decomposition set $\mathcal{DS}$ needs to include role successors of all *related roles* for a node v, which is defined as $\mathcal{R}(v) = \{R \,|\bowtie n\ R.A$ occurs in $\mathcal{L}(v)\}$. If a role R appears in a partition element p, $R \in p$, it means that all the elements of $p^{\mathcal{I}}$ should be R successors. If a role does not appear in p, the elements of $p^{\mathcal{I}}$ are assumed to be R' successors where R' is the complement of R.

One needs to include *qualifying concepts* in the decomposition set to capture the semantics of QCRs and also to handle their possible interaction with nominals. It allows to distinguish cases where role successors have different qualifications. When a partition element p includes a concept name A, it means that all the elements of $p^{\mathcal{I}}$ are elements of $A^{\mathcal{I}}$. Otherwise, they are assumed to be elements of $(\neg A)^{\mathcal{I}}$. The set of qualifying concepts of R related to node v is defined as $Q_v(R) = \{A \,|\bowtie nR.A$ occurs in $\mathcal{L}(v)\}$. Accordingly, the set of all qualifying concepts related to a node v is defined as $Q_v = \bigcup_{R \in R(v)} Q_v(R)$.

Having nominals as part of our decomposition set allows reflecting their semantics in inequalities. Besides, the interpretation of each nominal $o \in N_o$ may interact with successors of a role $R \in N_R$ if $o^{\mathcal{I}} \subseteq Succ(R)$. Additionally, The same nominal can interact with role successors of another role S. In this case, although R and S may not be related to each other according to the role hierarchy, their role successors may interact due to their interaction with the common nominal. If a nominal $o \in N_o$ is part of a partition element p, then it means that all the elements of $p^{\mathcal{I}}$ have to be in $o^{\mathcal{I}}$. The set of related nominals for a node v, $N_o(v) \subseteq N_o$, is defined as $N_o(v) = \{o \mid o$ occurs in $\mathcal{L}(v_C)$ such that $C \in Q_v\}$. If a nominal o is not part of p then its negation $\neg o$ is added to p by default. The restrictions imposed by nominals can not be handled node-locally. That is why nominals participate in the decomposition set of all related nodes.

Definition 8 (Decomposition Set). *A decomposition set $\mathcal{DS}(v)$ is defined for each node v, as $\mathcal{DS}(v) = \mathcal{R}(v) \cup Q_v \cup N_o(v)$.*

Definition 9 (Partition). *A partition $\mathcal{P}$ is the* power set *of $\mathcal{DS}(v) \cup N_{\neg o}$, where $N_{\neg o}$ is the set of the negation of all nominals occurring in $\mathcal{O}$. Each $p \in \mathcal{P}$ is associated with a variable σ_p, which is equal to the cardinality of $p^{\mathcal{I}}$.*

4.2 Deriving Inequalities

The partition elements $p \in \mathcal{P}$ are defined using the atomic decomposition technique, so they are semantically pairwise disjoint. Since the cardinality function of disjoint sets is additive, one can encode a cardinality restriction into an inequality using the sum of cardinalities, shown by δ_{sq}, such that for each $sq \subseteq \mathcal{DS}$ we have $\delta_{sq} = \Sigma_{sq \subseteq p} \sigma_p$ for all $p \in \mathcal{P}$. Roughly speaking, δ_{sq} is the sum of cardinalities of all partition elements containing sq. For example if $\mathcal{DS} = \{R, C, o\}$ then $\delta_R = \sigma_{RCo} + \sigma_{RC} + \sigma_{Ro} + \sigma_R$ and $\delta_{RC} = \sigma_{RCo} + \sigma_{RC}$. Hence, a cardinality bound imposed by QCRs or nominals on a subset of domain elements distributed over the elements in $\mathcal{P}$ can be encoded into inequalities as follows.

QCRs of the form $\leq nR.C$ or $\geq mS.D$ are mapped to inequalities $\delta_{RC} \leq n$ or $\delta_{SD} \geq m$ respectively. Based on nominal semantics, the sum of cardinalities of all partition elements containing a particular nominal should be equal to 1. We encode this cardinality bound by adding two inequalities of the form $\delta_o \leq 1$ and $\delta_o \geq 1$, for each nominal $o \in N_o$.

Providing more information to the ILP component reduces the risk that a solution is returned which will fail later due to newly derived entailments.

Table 1. Inference Rules for reasoning in $\mathcal{ALCHOQ}$

Subs	**If**	$\bigsqcap_{i=1}^{n} A_i \sqsubseteq M \in \mathcal{O}$,
		$K_i \sqsubseteq M_i \sqcup A_i \in \mathcal{L}(v)$
		and $\bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i \notin_* \mathcal{L}(v)$
	then	add $\bigsqcap_{i=1}^{n} K_i \sqsubseteq M \sqcup \bigsqcup_{i=1}^{n} M_i$ to $\mathcal{L}(v)$
Sigma	**If**	there exists p occurring in $\xi(v)$ such that $R \in p$ and
		no edge $\langle v, u\rangle \in \mathcal{E}$ exists
		such that $R \in \mathcal{L}\langle v, u\rangle$ and $L \sqsubseteq L \in_* \mathcal{L}(u)$ for each $L \in p$
	then	let $u = \texttt{select}(p)$
		if u does not exist,
		then create node u and $\mathcal{L}(u) = \{\top \sqsubseteq L \mid L \in \texttt{core}(u)\}$;
		if $\langle v, u\rangle$ does not exist,
		then create edge $\langle v, u\rangle$; and add R to $\mathcal{L}(\langle v, u\rangle)$; and
		for each $L \in p \backslash \texttt{core}(u)$ such that $L \sqsubseteq L \notin_* \mathcal{L}(u)$, add $L \sqsubseteq L$ to $\mathcal{L}(u)$.
Nom	**If**	there exists p occurring in $\xi(v)$ such that a nominal $o \in p$,
		and a concept/nominal $D \in p$ and,
		$\mathcal{L}(v) \cup \{o \sqcap D \sqsubseteq \bot\}$ causes a **strict** unsatisfiability and,
		$\top \sqsubseteq D \notin_* \mathcal{L}(o)$,
	then	add $\top \sqsubseteq D$ to $\mathcal{L}(o)$.
Bottom	**If**	there exists a node u such that $\texttt{core}(u) \sqcap K_i \sqsubseteq M_i \sqcup L_i \in CC_j(v)$ for $0 \leq i \leq n$,
		$CC_j(v) \in CC(v)$,
		where either $L_i \in \boldsymbol{Q}_p(v)$ or $\texttt{core}(u) \sqcap K_i \sqsubseteq M_i \sqcup L_i \in \bigcup_{o \in N_o} \boldsymbol{C}_p(o)$ and,
		$\bigsqcap_{i=0}^{n} K_i \sqsubseteq \bigsqcup_{i=0}^{n} M_i \notin_* \mathcal{L}(u)$,
	then	add $\bigsqcap_{i=0}^{n} K_i \sqsubseteq \bigsqcup_{i=0}^{n} M_i$ to $\mathcal{L}(u)$.

Feeding subsumptions and disjointness to the ILP component allows the so-called *infeasible* operator to set the cardinality of some partition elements to zero. For this purpose, we define a *binary variable* $b_C \in \{0, 1\}$, $C \in \mathcal{DS}$, associated with each member of a decomposition set in order to apply conditional constraints on the presence of a role, concept or nominal in a partition element. Assuming that $b_\bot \leq 0$, we use binary variables and inequalities to disable infeasible partition elements, using the mappings presented below.

Subsumption Relation Assuming that in a clause of the form $K \sqsubseteq M$, where K denotes $\bigsqcap_{i=1}^{n} L_i$ and M denotes $\bigsqcup_{j=1}^{m} L_j$, all the literals are atomic concepts or nominals and $L_i \in \mathcal{DS}$. Then the ILP component maps clause $K \sqsubseteq M$ to the inequality $\sum_{i=1}^{n} b_{L_i} - (n-1) \leq \sum_{j=1}^{m} b_{L_j}$. For example translating a subsumption relation $A \sqsubseteq B$ produces $b_A \leq b_B$, which ensures that if a partition element contains the atomic concept A, it should contain A's subsumer B. In other words the cardinality of a partition element which contains A but not B has to be equal to zero.

This mapping can also be used for encoding axioms of the form $A \sqsubseteq o_1 \sqcup o_2$ where o_1 and o_2 are two nominals. The obtained inequality, $b_A \leq b_{o_1} + b_{o_2}$, guarantees that if a partition element contains the atomic concept A then it has to contain either o_1 or o_2, i.e., the cardinality of a partition element that contains A but neither o_1 nor o_2, should be equal to zero.

Role Subsumption Every role subsumption relation of the form $R \sqsubseteq S$ can be encoded to the inequality $b_R \leq b_S$. This inequality ensures that if a partition

element contains a subrole R, it must contain all of R's superroles, otherwise the cardinality of such a partition element should be equal to zero.

Universal Restriction A universal restriction of the form $\forall R.A$ can be encoded to the inequality $b_R \leq b_A$. The semantics of universal restriction implies that all R successors are instance of A. The generated inequality satisfies this semantics by implying that if a partition element contains the role R it should also contain the concept A.

4.3 Returning Solution or Clash Culprits

Lemma 1. *If the submitted inequality system to the ILP component is feasible, it will return a solution that satisfies all the constraints. The returned arithmetic solution assigns a positive integer n to σ_p, which denotes the cardinality of the corresponding partition element. The ILP component would iterative through all possible solutions that exist for solving an equality system [3].*

Definition 10 (Arithmetic Solution). *An arithmetic solution $\xi(v)$ is a set of tuples $\langle p, \sigma_p \rangle$ produced by the ILP component for solving a particular inequality system related to a node v, where p is a partition element and $\sigma_p \in \mathbb{N}, \sigma_p \geq 1$ is the cardinality of p.*

If the inequality system is infeasible, then the ILP component returns the smallest set of clauses, called *clash culprit set* (Definition 7), such that the inequality system containing their corresponding inequalities is infeasible. There may be more than one minimum clash culprit set for each inequality system.

5 Summary and Conclusion

This paper presented an algebraic consequence-based reasoning algorithm for $\mathcal{SHOQ}$. To the best of our knowledge, this is the first extension of consequence-based algorithms for the expressive description logic $\mathcal{SHOQ}$. During the reasoning process, only one node is created for representing elements associated with each concept. Using a representative node for each concept not only helps in reducing the size of the generated framework but also allows for re-using elements. In contrast to tableau-based reasoners, our calculus naturally handles cyclic descriptions without the need for any blocking strategies to ensure termination.

Unlike most reasoning algorithms for $\mathcal{SHOQ}$, the algebraic consequence-based method allows arithmetically informed reasoning about the numerical restrictions on domain elements by mapping numerical restrictions to inequalities and handling obtained inequality systems using integer linear programming methods. The consequence-based $\mathcal{SHOQ}$ reasoning is based on the atomic decomposition technique which is applied to the proper decomposition set allowing the calculus to handle the various interactions between nominals, role successors and their qualifications. The inference rules are designed in a way to derive all consequences of presented axioms while benefiting from *lazy unfolding* to avoid overwhelming the framework with unnecessary axioms. The inference rules are inspired by resolution to resolve complement literals.

References

1. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 364–369, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.
2. A. Bate, B. Motik, B. C. Grau, F. Simancik, and I. Horrocks. Extending consequence-based reasoning to $\mathcal{SRIQ}$. In C. Baral, J. P. Delgrande, and F. Wolter, editors, *KR*, pages 187–196. AAAI Press, 2016.
3. J. Faddoul. *Reasoning Algebraically with Description Logics.* PhD thesis, Concordia University, Montreal, Quebec, Canada, September 2011.
4. J. Faddoul and V. Haarslev. Algebraic Tableau Reasoning for the Description Logic $\mathcal{SHOQ}$. *Journal of Applied Logic, Special Issue on Hybrid Logics*, 8(4):334–355, December 2010.
5. V. Haarslev and R. Möller. Racer system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.
6. Y. Kazakov. Consequence-driven reasoning for Horn-$\mathcal{SHIQ}$ ontologies. In C. Boutilier, editor, *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009*, pages 2040–2045, 2009.
7. Y. Kazakov, M. Krötzsch, and F. Simančík. Practical reasoning with nominals in the $\mathcal{EL}$ family of description logics. In G. Brewka, T. Eiter, and S. A. McIlraith, editors, *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*, pages 264–274. AAAI Press, 2012.
8. B. Motik, R. Shearer, and I. Horrocks. Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.
9. J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
10. F. Simančík. *Consequence-Based Reasoning for Ontology Classification.* PhD thesis, University of Oxford, 2013.
11. F. Simančík, Y. Kazakov, and I. Horrocks. Consequence-based reasoning beyond Horn ontologies. In T. Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 1093–1098. AAAI Press/IJCAI, 2011.
12. F. Simančík, B. Motik, and I. Horrocks. Consequence-based and fixed-parameter tractable reasoning in description logics. *Artificial Intelligence*, 209:29–77, 2014.
13. E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.
14. D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.