Querying Guarded Fragments via Resolution

Sen Zheng  
Department of Computer Science  
University of Manchester

Renate A. Schmidt  
Department of Computer Science  
University of Manchester

Abstract
The problem of answering Boolean conjunctive queries over the guarded fragment is decidable, however, as yet no practical decision procedure exists. In this paper, we present a resolution-based decision procedure to address this problem, and further extend the decision procedure to the loosely guarded fragment. In particular, we show that using a separation rule and top-variable-based techniques, one can rewrite Boolean conjunctive queries into a set of (loosely) guarded clauses, so that querying the (loosely) guarded fragment can be reduced to deciding the (loosely) guarded fragment. As far as we know, this is the first practical decision procedure for answering Boolean conjunctive queries over the guarded fragment and the loosely guarded fragment.

1 Introduction
Answering queries over knowledge bases is at the heart of knowledge representation research. In this paper, we are interested in the problem of answering Boolean conjunctive queries. A Boolean conjunctive query (BCQ) is a first-order formula of the form \( q = \exists \varphi(\vec{x}) \) where \( \varphi \) is a conjunction of atoms containing only constants and variables. Given a Boolean conjunctive query \( q \), a set of rules \( \Sigma \) and a database \( D \), the aim is to check whether \( \Sigma \cup D \models q \). The BCQ answering problem can be formed as a vital problem in many research areas, such as query evaluation, query entailment [5] and query containment in database research [11], and constraint-satisfaction problem and homomorphism problems in general AI research [38].

In this paper, we consider the case where the rules \( \Sigma \) are in the guarded fragment [2] and the loosely guarded fragment [36]. Formulas in the guarded fragment (GF) are first-order formulas, in which the quantification is restricted to the form \( \exists \exists(G(\vec{x}) \land \varphi(\vec{x}, \vec{y})) \) where the atom \( G \) (called guard) contains all free variables of \( \varphi \). Satisfiability in many decidable propositional modal logics such as \( K, D, S3 \) and \( B \) can be encoded as satisfiability of formulas in GF. The loosely guarded fragment (LGF) further extends GF by allowing multiple guards, so that for instance, LGF can express the until operator in temporal logic. Both GF and LGF inherit robust decidability—the tree model property [37] from modal logic [22, 25], and have been investigated from theoretical perspective [2, 21, 22] and automated reasoning [12, 15, 24].

Due to these good properties, querying in GF and LGF has attracted attention in general database research and research on ontology-based data access (OBDA) systems [9]. In the database research, it has been shown that BCQs \( q \) in the (loosely) guarded fragment that have the (hyper) tree width property [20] can be evaluated over a database \( D \) (checking \( D \models q \)) in LOGCFL time [19], an even better complexity result than polynomial-time. On the other hand, in an OBDA system, the description logic \( \mathcal{ALCHOIQ} \) and its fragments [10, 29, 30, 34], and guarded existential rules (a.k.a. guard tuple-generating dependency, guarded Datalog\(^{+/-}\)) [8] are commonly used as ontological languages. The former can be obtained by limiting arities of predicate symbols and the number of variables in guarded formulas, and the latter, guarded existential rules, syntactically are Horn guarded formulas.

BCQ answering for GF is \( 2\text{ExpTime} \)-complete [6], and satisfiability checking for the clique-guarded negation fragment, which can be reduced to the problem of BCQ answering against LGF, is also \( 2\text{ExpTime} \)-complete [7]. These results mean that BCQ answering for both GF and LGF are decidable, however, as yet there are no practical decision procedures.

In this paper, we present a resolution-based decision procedure to solve BCQ answering problems in GF and LGF. Resolution provides a powerful method for developing practical decision procedures as has been shown in [4, 12, 15, 16, 18, 26–28]. The BCQ answering problem is equivalent to satisfiability problem of \( \Sigma \cup D \cup \neg q \), where \( q \) is a BCQ. \( D \) is a set of ground atoms and \( \Sigma \) are formulas in the (loosely) guarded fragment. Figure 1 illustrates the basic steps of how to decide satisfiability of \( \Sigma \cup D \cup \neg q \):

1. Transform the (loosely) guarded fragment and BCQs into their clausal forms, denoted as the (loosely) guarded clauses and query clauses, respectively.
2. Apply our refined-resolution-based inference system to the (loosely) guarded clauses and query clauses.

We prove that each logical consequence derived from our system has bounded depth and length, thereby the system guarantees termination. Also, we proved that our system is sound and refutationally complete.

Figure 1. Basic steps of a resolution-based procedure
In this paper, one of the main challenges is handling query clauses, since these clauses go beyond guarded clauses and loosely guarded clauses. By simply negating a BCQ, one can obtain a query clause that is a negative clause containing only variables and constants. An example of query clause is $Q = \neg A_1 x y \vee \neg A_2 y z$. We use a variation of the separation rule, called Sep, to rewrite $Q$ into guarded clauses. The separation rule is used as a dynamic renaming technique to decide fluted logic [35]. The idea behind Sep is to separate the literals that share common variables in query clauses. For example, using Sep, $Q$ can be separated into: $C_1 = \neg A_1 x y \vee q y z$ and $C_2 = \neg q y \vee \neg A_2 y z$ where $q$ is a fresh predicate symbol. Both $C_1$ and $C_2$ are guarded clauses. In fact, we discovered that our query rewriting based on Sep and the splitting rule [3] behaves like the GYO-reduction [40], where cyclic queries [39] are identified by recursively removing 'ears' in the hypergraph of $Q$. In Lemma 5, we show that using Sep and the splitting rule, we can always rewrite an acyclic query clause into a set of guarded clauses. However, for cyclic query clauses such as $Q = \neg A_1 x y \vee \neg A_2 y z \vee \neg A_3 x z$, since $Q$ contains only common variables, Sep and the splitting rule are not sufficient to express $Q$ using guarded clauses.

For cyclic query clauses, we use resolution to break the cycles. Some caution is needed since applying unrefined resolution can easily increase the depth of the resolvents.

**Example 1.** Given $Q$ and a set of guarded clauses $C_1, C_2, C_3$:

\[
\begin{align*}
Q & = \neg A_1 x y \vee \neg A_2 y z \vee \neg A_3 x z \\
C_1 & = A_1 (f x, x) \vee D(g x) \vee \neg G_1 x \\
C_2 & = A_2 (f x, f x) \vee \neg G_2 x \\
C_3 & = A_3 (x, f x) \vee \neg G_3 x
\end{align*}
\]

using $Q, C_1, C_2$ and $C_3$, one can derive the resolvent $D(q(f x)) \vee \neg G_1 (f x) \vee \neg G_2 x \vee \neg G_3 x$, in which the deeper term $g(f x)$ occurs. We use the top variable technique [12, 15, 41], which only allows inferences on the literals that contain the potentially deepest terms, to avoid such term depth increase.

In Example 1, the top variable technique first finds the simultaneous most general unifier $\sigma = \{x/f f x', y/f x', z/f x'\}$ for variables in $Q$; $x$ is identified as the top variable and resolution is performed on the literals containing $x$ in $Q$. This means top-variable-based resolution is performed only on $Q, C_1$ and $C_3$, deriving $\neg A_2 x x \vee D(g x) \vee \neg G_1 x \vee \neg G_3 x$, where no term depth increases.

The idea of resolving the literals that contain the potentially deepest terms was introduced as the ‘MAXVAR’ technique to decide LGF [12, 15]. In [41], this idea was further generalised to include queries, and formally shown to decide BCQs answering problems over the Horn LGF. The essence of this result is due to the property of Horn clauses, resolvents derived by using the top variable technique are query clauses. However, this is not the case for the whole of LGF, or even GF. Example 5 in Section 5.2 is a counter example showing that resolvents derived by using the top variable technique may be neither (loosely) guarded clauses nor query clauses. When we look at the whole of GF and LGF, Example 5 shows that merely using the top variable technique can cause the length of resolvents to grow, even though top-variable-based resolution can avoid term depth increase in the resolvents. We show that such a length increase can be avoided by applying a dynamic renaming technique, referred to as T-Trans, which transforms a resolvent obtained by using the top variable technique, into a query clause and a set of (loosely) guarded clauses. We further show that T-Trans not only can rewrite cyclic query clauses into the (loosely) guarded clauses, but also guarantees termination. In Section 5, we show that by reasonably applying a separation rule, the splitting rule, the top-variable-based resolution and T-Trans, query clauses can be effectively rewritten into a set of (loosely) guarded clauses. We refer to such a query rewriting procedure as Q-Proc. Having a set of (loosely) guarded clauses, another main challenge is building an inference system to reason with these clauses. Existing inference systems for GF and LGF are either based on tableau (see [23, 24]) or resolution (see [12, 15, 41]). Since our aim is to develop an inference system in line with a standard framework [3], we develop our system as a variation of [15, 41], which are the only existing systems tend to decide GF and LGF that use the framework [3], to take advantage of simplification rules and notions of redundancy elimination. In particular, our inference system can be seamlessly combined with Q-Proc, giving us as a query answering system for answering BCQs for GF and LGF.

The main contributions of this paper are:

- A practical query rewriting procedure Q-Proc, which as far as we know, is the first procedure that can rewrite query clauses into (loosely) guarded clauses.
- By combining Q-Proc with our inference system for GF and LGF, we present the first, as far as we know, practical decision procedure to solve BCQ answering problems for GF and LGF, together with soundness and completeness proofs.

### 2 Preliminaries

Let $C, F, P$ denote pairwise disjoint discrete sets of constant symbols, function symbols and predicate symbols, respectively. A **term** is either a variable or an expression $f(t_1, \ldots, t_n)$ where $f$ is a function symbol of arity $n$ and $t_1, \ldots, t_n$ are terms. A **compound term** is a term that is neither a variable nor a constant. A **ground term** is a term containing no variables. An atom is an expression $P(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms and $P$ is an $n$-ary predicate symbol in $P$. A literal is an atom $A$ (a positive literal) or a negated atom $\neg A$ (a negative literal). For a literal $L = (\neg)P(t_1, \ldots, t_n)$ the terms $t_1, \ldots, t_n$ are the arguments of $L$. A literal $L$ is a non-ground compound literal if $L$ contains at least one non-ground compound term. A first-order clause is a multiset of literals. An expression can
be a term, an atom, a literal, or a clause. Given two expressions \(A, \ldots, t, \ldots\) and \(B, \ldots, u, \ldots\), we say \(t\) matches \(u\) if the argument position of \(t\) in \(A\) is the same as the argument position of \(u\) in \(B\).

A substitution is a mapping defined on variables, where variables denoting terms are mapped to terms and variables denoting formulas, to formulas. By \(E\sigma\) we denote the result of applying the substitution \(\sigma\) to an expression \(E\) and call \(E\sigma\) an instance of \(E\). An expression \(E'\) is a variant of an expression \(E\) if there exists a variable substitution \(\sigma\) such that \(E'\sigma = E\sigma\). Two substitutions are equal, denoted \(\sigma = \theta\), if \(x\sigma = x\theta\) for every variable \(x\). We say that \(\sigma\) is more general than \(\theta\), denoted as \(\sigma <_m \theta\), if there exists an \(\eta\) such that \(\theta = \sigma\eta\).

A substitution \(\sigma\) is a unifier of two terms \(s\) and \(t\) if \(s\sigma = t\sigma\); it is a most general unifier (mgu), if for every unifier \(\theta\) of \(s\) and \(t\), \(\sigma <_m \theta\). The simultaneous most general unifier (simultaneous mgu) \(\sigma\) is the mgu of two sequences of terms \(s_1, \ldots, s_n\) and \(t_1, \ldots, t_n\) such that \(s_i\sigma = t_i\sigma\) for each \(1 \leq i \leq n\). In this paper, we abusively use the notion mgu to denote the notion of simultaneous mgu.

A compound term \(t\) is weakly covering [14] if for every non-ground, compound subterm \(s\) of \(t\), it is the case that \(\text{var}(s) = \text{var}(t)\). A literal \(L\) is weakly covering if each argument of \(L\) is either a ground term, a variable, or a weakly covering term \(t\) such that \(\text{var}(t) = \text{var}(L)\). A clause \(C\) is weakly covering if each term \(t\) in \(C\) is either a ground term, a variable, or a weakly covering term such that \(\text{var}(t) = \text{var}(C)\).

We use \(\text{vdpt}(t)\) to denote the variable depth of a term \(t\), formally defined as:

- if \(t\) is ground, then \(\text{vdpt}(t) = -1\),
- if \(t\) is a variable, then \(\text{vdpt}(t) = 0\), and
- if \(t\) is a non-ground compound term \(f(u_1, \ldots, u_n)\), then \(\text{vdpt}(t) = 1 + \max\{\text{vdpt}(u_i) \mid 1 \leq i \leq n\}\).

A term \(t\) is flat if \(\text{vdpt}(t) \leq 0\). A term \(t\) is simple if \(\text{vdpt}(t) \leq 1\). A flat (simple) atom, literal, and clause is an atom, a literal and a clause such that every term in it is flat (simple).

Let \(\overline{x}, \overline{A}, \overline{C}\) denote a sequence of variables, a sequence of atoms, and a set of clauses, respectively. Let \(\text{var}(t), \text{var}(C)\) and \(\text{var}(\overline{A}_n)\) be sets of variables in a term \(t\), a clause \(C\) and a sequence of atoms \(\overline{A}_n\), respectively.

The rule set \(\Sigma\) denotes a set of first-order formulas and the database \(\mathcal{D}\) denotes a set of ground atoms. A Boolean conjunctive query is a first-order formula of the form \(q = \exists \Phi(\overline{x})\) where \(\Phi\) is a conjunction of atoms containing only constants and variables. We use the symbol \(Q\) to denote the query clause \(\neg q\). Thus we can answer a Boolean conjunctive query \(\Sigma \cup \mathcal{D} \models q\) by checking whether \(\Sigma \cup \mathcal{D} \cup Q \models \bot\).

In this work, we particularly focus on the case where \(\Sigma\) is expressed in GF or LGF.

3 From Logic Fragments to Clausal Sets

In this section, we first provide formal definitions of GF and LGF, and introduce a specially optimised structural transformation such that guarded formulas, loosely guarded formulas and BCQs can be converted into suitable sets of clauses.

We first give the definition of the guarded fragment:

Definition 1 (Guarded Fragment). Without equality and function symbols, the guarded fragment (GF) is a fragment first-order logic defined inductively as follows:

1. \(\top\) and \(\bot\) belong to GF.
2. If \(A\) is an atom, then \(A\) belongs to GF.
3. GF is closed under Boolean combinations.
4. If \(F\) is a formula in GF and \(G\) is an atom, then a formula \(\forall x(G \rightarrow F)\) belongs to GF if all free variables of \(F\) belong to \(\text{var}(G)\). \(G\) is denoted as the guard of \(\forall x(G \rightarrow F)\).

In GF, each quantified formula contains a guard literal.

Example 2. Some (counter) examples of guarded formulas:

\[
\begin{align*}
F_1 &= Ax \\
F_2 &= \forall x Ax \\
F_3 &= \forall x(Axy \rightarrow Bxy) \\
F_4 &= \forall x(Axy \rightarrow \exists yByz) \\
F_5 &= \forall x(Axy \rightarrow \bot) \\
F_6 &= \exists x(Axy \land \forall z(Bxz \rightarrow \exists wCzw)) \\
F_7 &= Px \rightarrow \exists y(Rxy \land \forall z(Ryz \rightarrow Pz))
\end{align*}
\]

Formulas 1, 3, 5, 6, 7 are guarded formulas, and the rest are not. Formula 2, 4 are not guarded formulas because they do not contain ‘guard’ literals. Formula 6 is the standard first-order translation of the modal formula \(P \rightarrow \Box \Diamond P\) (with respect to one world) and the description logic axiom \(P \subseteq \exists R.\forall Y.\forall P\).

Now we give the formal definition of LGF. LGF further extends GF by relaxing Condition 4 in the definition of GF.

Definition 2 (Loosely Guarded Fragment). Without equality and function symbols, the loosely guarded fragment (LGF) is a fragment of first-order logic defined inductively as follows:

1. \(\top\) and \(\bot\) belong to LGF.
2. If \(A\) is an atom, then \(A\) belongs to LGF.
3. LGF is closed under Boolean combinations.
4. If \(F\) is a formula in LGF and \(G_1, \ldots, G_n\) are atoms, then a formula \(\forall x(G_1 \land \ldots \land G_n \rightarrow F)\) belongs to LGF if i) all free variables of \(F\) belong to \(\text{var}(G_1, \ldots, G_n)\), and ii) for each variable \(y\) in \(\overline{x}\) and each variable \(y\) in \(\text{var}(G_1, \ldots, G_n)\) where \(x \neq y\), \(x\) and \(y\) co-occur in a
$G_1$. The negative literals $\neg G_1, \ldots, \neg G_n$ are called the guards of $\forall x(G_1 \land \ldots \land G_n \rightarrow F)$. \[ \]

LGF strictly extends GF by allowing conjunction of guards in the guard position. E.g., $\forall z(Rxz \land Rzy) \rightarrow Pz$ belongs to LGF, but does not belong to GF. The first-order translation of a temporal logic formula $P$ until $Q$ is a loosely guarded formula $\exists y(Rxy \land Qy \land \exists z((Rxz \land Rzy) \rightarrow Pz))$, but not guarded, and the transitivity formula $\forall x y z ((Rxy \land Rzy) \rightarrow Rxz)$ is neither guarded nor loosely guarded.

**Clausal Transformation** We now introduce the clausal transformation for GF/LGF and BCQs. We use $G$-Trans to denote our clausal transformation, which is a variation of the structural transformation in [12, 15, 41].

Notice that we explicitly assume that all free variables are existential quantified, and outer Skolemisation [32] is used to eliminate existential quantifications. If $\forall x \exists y \phi$ is a first order formula where $\phi$ is quantifier free and $\forall$ are free variables in $\exists y \phi$, then $\phi[y/f(\exists \forall)]$ is the outer Skolemisation, while $\phi[y/g(\exists \forall)]$ is the standard/inner Skolemisation. E.g., given a formula $\forall x z (Rxz \lor \exists y (Pxy))$, $\forall x z (Rxz \lor P(x, gx))$ is the result of inner/standard Skolemisation and $\forall x z (Rxz \lor P(x, f(xz)))$ is the result of outer Skolemisation where $g$ and $f$ are Skolem functions. Although outer Skolemisation generally introduces Skolem functions with higher arity than inner/standard Skolemisation, for us, outer Skolemisation is essential to guarantee that the obtained clauses are weakly covering.

Let us take the guarded formula $F_6 = \exists x (Axy \land \exists z (Bxz \rightarrow \exists u Czu))$ from Example 2. Using $G$-Trans, $F_6$ can be transformed into a set of clauses as follows:

1. Add existential quantifiers for all free variables in $F_6$.
   $\exists y \exists x (Axy \land \exists z (Bxz \rightarrow \exists u Czu))$

2. Rewrite $\rightarrow$ and $\Leftarrow$ using conjunctions, disjunctions and negations, and transform $F$ into negation normal form, obtaining the formula $F_{\text{nnf}}$.
   $\exists y \exists x (Axy \land \exists z (\neg Bxz \lor \exists u Czu))$

3. Apply the optimised structural transformation: introduce fresh predicate symbols $Q_i$ for universally quantified subformulas (with quantifiers), obtaining $F_{\text{str}}$.
   We call such a predicate symbol $Q_i$ a $\forall$-definer.
   $\exists y \exists x (Axy \land Q_i x)$, $\forall x (\neg Q_i x \lor \forall z (\neg Bxz \lor \exists u Czu))$

4. Transform formulas in $F_{\text{str}}$ into prenex normal form and use outer Skolemisation, introducing Skolem constants $a$ and $b$ and a Skolem function $f$, obtaining $F_{\text{sko}}$.
   $Aab \land Q_i a$, $\forall x (\neg Q_i x \lor \neg Bxz \lor C(z, f(xz)))$

5. Drop all universal quantifiers and transform $F_{\text{sko}}$ into conjunctive normal form, obtaining these clauses:
   $Aab$, $Q_i a$, $\neg Q_i x \lor \neg Bxz \lor C(z, f(xz))$

We use $B$-Trans to denote the clausal transformation for BCQs: Simply negate the BCQ to obtain a query clause. Hence, query clauses can be defined as:

**Definition 3.** A query clause is a negative clause containing only variables and constants.

The clausal normal form transformation $G$-Trans converts (loosely) guarded formulas into (loosely) guarded clauses, which we deliver next:

**Definition 4 (Guarded Clause).** A guarded clause (GC) $C$ is a first-order clause that satisfies the following conditions:

1. $C$ is either ground, or
2. $C$ is simple and weakly covering, and
3. contains at least one negative flat literal $\neg G$ such that $\text{var}(C) = \text{var}(G)$.

Notice that using $G$-Trans, a non-guarded formula can also be transformed into a guarded clause. E.g., $\forall x (Ax \rightarrow D(fx))$ is not a guarded formula, but its clausal normal form $\neg Ax \lor D(fx)$ is a guarded clause. A guarded clause is not always a query clause, and vice versa. E.g., $\neg Ax y \lor \neg Bx y$ is a guarded clause and a query clause. However, $\neg Ax y \lor B(fxy)$ is a guarded clause but not a query clause, and $\neg Ax y \lor \neg Byz$ is a query clause, but not a guarded clause.

**Definition 5 (Loosely Guarded Clause).** A loosely guarded clause (LGC) $C$ is a first-order clause satisfying the following conditions:

1. $C$ is either ground, or
2. $C$ is simple and weakly covering, and
3. there is a set of negative literals $\neg G_1, \ldots, \neg G_n$ in $C$ that are flat, and each pair of variables in $C$ co-occur in at least one of literals in $\neg G_1, \ldots, \neg G_n$.

One can see that the notion of an LGC is a strict extension of that of GCs because Condition 3 in the definition of LGC allows multiple guards.

**Example 3.** Some (counter) examples of LGCs:

- $C_1 = \neg A_1 xy \lor \neg A_2 yz \lor \neg A_3 zx$
- $C_2 = \neg A_1 xy u \lor \neg A_2 yzu \lor \neg A_3 zxw$
- $C_3 = \neg A_1 xy \lor \neg A_2 yz \lor \neg A_3 zu$

$C_1$ is an LGC, but $C_2$ and $C_3$ are not because variables in them do not co-occur in at least one negative flat literals of these clauses. Note that $C_1$, $C_2$ and $C_3$ are all query clauses, and $C_1$ is the cyclic query clause in Example 1 in Introduction.

**Proposition 1.** Using $G$-Trans, every (loosely) guarded formula can be transformed into a set of (loosely) guarded clauses, and such procedure is $\text{Ptime}$-hard.

**Proof.** This proof explains how a (loosely) guarded formula is transformed to be a set of clauses satisfying conditions in the definition of guarded clauses and loosely guarded clauses.

Let $C$ be a non-ground (loosely) guarded clause. $C$ is simple because a (loosely) guarded formula contains no non-ground compound term, and all non-ground compound terms in $C$ are Skolem function terms. Prenex normal form and outer
Skolemisation guarantees that every non-ground compound terms in \( C \) contains all variables of \( C \); thus \( C \) is weakly covering. For Condition 3 in the definition of GF and LGF, let a (loosely) guarded formula \( F \) is of form \( \forall \varphi(G_1, \ldots, G_n \rightarrow F_1) \) and \( F_1 \) is a (loosely) guarded formula. \( F \) is loosely guarded if \( n > 1 \). Variables in \( \neg G_1, \ldots, \neg G_n \) can either be bounded or free. According to Condition 4 of the definition of LGF, each bounded \( x \) co-occurs with a variable \( y \) that is either free or bounded in a \( \neg G_i \) in \( \neg G_1, \ldots, \neg G_n \). Each pair of free variables in \( \neg G_1, \ldots, \neg G_n \) co-occur in a \( \forall \)-definer \( Q_j \). Therefore, each pair of variables in \( C \) co-occur in at least a negative flat literals (\( \neg G_i \) and \( \neg Q_j \)) in \( C \). If \( n = 1 \), each pair of variables in \( C \) co-occur in one \( \neg G_i \). Hence, Condition 3 in the definition of GF and LGF holds.

As for the complexity of such transformation, if one introduces additional definers for every conjunctive formulas, such transformation can be done in \( \text{PTime} \). □

4 Top-Variable-based Inference System

In this section, we present the top-variable-based inference system from [41] (inspired by [12]), enhanced with the splitting rule and a new Lemma 1, to extend the system’s applicability. The system is defined in the spirit of [3] and provides a decision procedure for LGF and querying Horn LGF [41]. Based on this system, we build a decision procedure for querying the whole of GF and LGF.

Let > be a strict ordering, called a precedence, on the symbols in \( C, F \) and \( P \). An ordering \( > \) is liftable if \( E_1 > E_2 \) implies \( E_1 \sigma > E_2 \sigma \) for all expressions \( E_1, E_2 \) and all substitutions \( \sigma \). An ordering \( > \) on literals is admissible, if i) it is well-founded and total on ground literals, and liftable, ii) \( \neg A > A \) for all ground atoms \( A \), and iii) if \( B > A \), then \( B > \neg A \) for all ground atoms \( A \) and \( B \).

A ground literal \( L \) is \( > \)-maximal with respect to a ground clause \( C \) if for any \( L' \) in \( L \), \( L' > L \), and \( L \) is \( > \)-maximal with respect to \( C \) if for any \( L' \) in \( C \), \( L > L' \). A non-ground literal \( L \) is (strictly) maximal with respect to a non-ground clause \( C \) if and only if there is a ground substitution \( \sigma \) such that \( L \sigma \) is (strictly) maximal with respect to \( C \sigma \) such that for all \( L' \) in \( C \), \( L \sigma > L' \sigma \). A selection function \( S \) selects a possibly empty set of occurrences of negative literals in a clause \( C \) with no restriction imposed. Inferences are only performed on eligible literals. A literal \( L \) is eligible in a clause \( C \) if either nothing is selected by the selection function \( S \) and \( L \) is a \( > \)-maximal literal with respect to \( C \), or \( L \) is selected by \( S \).

A ground clause \( C \) is redundant with respect to \( N \) if there are ground instances \( C_1 \sigma, \ldots, C_n \sigma \) of clauses in \( N \) such that \( C_i \sigma, \ldots, C_n \sigma \models C \) and for each \( i, C > C_i \sigma \). A non-ground clauses \( C \) is redundant with respect to \( N \) if every ground instance of \( C \) is redundant with respect to \( N \). An inference is redundant if one of the premises is redundant, or its conclusion is redundant or an element of \( N \). A set of clauses \( N \) is saturated up to redundancy (with respect to ordered resolution and selection) if any inference from non-redundant premises in \( N \) is redundant in \( N \).

The top-variable-based inference system contains following rules:

- **Deduct:**
  \[
  \frac{N}{N \cup \{ C \}}
  \]
  if \( C \) is either a resolvent or a factor of clauses in the set \( N \).

- **Split:**
  \[
  \frac{N \cup \{ C \} \setminus \{ E \}}{N \cup \{ C \} \setminus \{ E \} \cup \{ D \}}
  \]
  if \( C \) and \( D \) are variable disjoint. A clause \( C \) is indecomposable if for any two subclasses \( C_1 \) and \( C_2 \) of \( C \), \( C_1 \) and \( C_2 \) are not variable disjoint. Hence, Split does not apply to indecomposable clauses.

- **Fact:**
  \[
  \frac{C \lor A_1 \lor A_2}{\frac{C \lor A_1 \lor A_2}{(C \lor A_1) \sigma}}
  \]
  where i) \( \sigma \) is an mgu of \( A_1 \) and \( A_2 \), and ii) no literal is selected in \( C \), and iii) \( A_1 \sigma \) is \( > \)-maximal with respect to \( C \sigma \).

- **Res:**
  \[
  \begin{align*}
  &\frac{B_1 \lor D_1 \ldots B_n \lor D_n}{\frac{\neg A_1 \lor \ldots \lor \neg A_n \lor D}{(D_1 \lor \ldots \lor D_n) \sigma}}
  
  \end{align*}
  \]
  where i) \( \neg A_1 \lor \ldots \lor \neg A_n \lor D, B_1 \lor D_1, \ldots, B_n \lor D_n \) are pairwise variable-disjoint, and ii) \( \sigma \) is an mgu such that \( B_1 \sigma \lor A_1 \sigma \) for all \( i \) such that \( 1 \leq i \leq n \), and iii) \( n > 1 \) and \( \neg A_1 \lor \ldots \lor \neg A_n \) are selected, and iv) no literal is selected in \( D_1, \ldots, D_n \) and \( B _1 \sigma, \ldots, B_n \sigma \) are strictly \( > \)-maximal with respect to \( D_1 \sigma, \ldots, D_n \sigma \), respectively.

- **BRes:**
  \[
  \frac{B_1 \lor D_1}{\frac{\neg A_1 \lor D}{(D_1 \lor D) \sigma}}
  \]
  where i) \( \neg A_1 \lor D, B_1 \lor D_1 \) are pairwise variable-disjoint, and ii) \( \sigma \) is an mgu such that \( B_1 \sigma = A_1 \sigma \), and iii) either \( A_1 \) is selected in \( \neg A_1 \lor D \), or nothing is selected in \( \neg A_1 \lor D \) and \( \neg A_1 \sigma \) is the maximal literal in \( \neg A_1 \lor D \sigma \), and iv) \( B_1 \sigma \) is strictly \( > \)-maximal with respect to \( D_1 \sigma \). BRes is the case when premises in Res are binary.

- **Cond:**
  \[
  \frac{N \cup \{ C \}}{N \cup \{ \text{Cond}(C) \}}
  \]
  if \( \text{Cond}(C) \) is a minimal subclause and also an instance of \( C \). A clause \( C \) is condensed if there is no condensation of \( C \) which is a strict subclause of \( C \).

- **Del:**
  \[
  \frac{N \cup \{ C \}}{N}
  \]
  if \( C \) is a tautology, or \( N \) contains a clause as a variant of \( C \).
In an application of TRes (at the top of next page), given query pair clauses (Definition 3 in [41]) $B_1 \lor D_1, \ldots, B_t \lor D_t$ and $\neg A_1 \lor \ldots \lor \neg A_n \lor D$, the top variables are variables in the main premise $\neg A_1 \lor \ldots \lor \neg A_n \lor D$ that are unified to terms having the deepest variable depth in the resolvent.

An application of TRes consists of three steps:

1. Without producing or adding the resolvent, perform Res among $\neg A_1 \lor \ldots \lor \neg A_n \lor D$ and $B_1 \lor D_1, \ldots, B_t \lor D_t$ to compute the mgu $\sigma'$ such that $B_i \sigma' = A_i \sigma'$ for each $i$ such that $1 \leq i \leq t$.

2. Compute the variable order among all pairs of variables $x$ and $y$ in $\neg A_1 \lor \ldots \lor \neg A_n$ using ordering $>_d$ defined by: $x >_d y$ if $\text{vdp}(x\sigma') > \text{vdp}(y\sigma')$, and $x =_d y$ if $\text{vdp}(x\sigma') = \text{vdp}(y\sigma')$.

3. Based on $>_d$ and $=_d$, identify the maximal variables, denoted as the top variables, among variables in $\neg A_1 \lor \ldots \lor \neg A_n$. In TRes, we use $\neg A_1 \lor \ldots \lor \neg A_t$ to denote literals in $\neg A_1 \lor \ldots \lor \neg A_n$ that contain top variables.

Del and Cond are the only rules used to eliminate redundancy. Since this paper focuses mainly on the termination result, they are sufficient. The resolution refinement we employ is based on admissible orderings with selection function, in accordance with the framework of [3]. This implies that more sophisticated simplification rules and redundancy elimination can be used, e.g., subsumption elimination, forward subsumption and backward subsumption [3].

Resolution Refinement

The refinement T-Refine we use for our calculus is described in Algorithm 1, which is based on a lexicographic path ordering $>_\text{lp}$ [13] with a precedence $f > c > p$ for $f \in F, c \in C$ and $p \in P$, and selection functions.

| Algorithm 1: The resolution refinement T-Refine |
|-----------------------------------------------|
| **Input:** A first-order clause $C$ |
| **Output:** Eligible literals $L$ in $C$ |
| 1 if $C$ is ground then $L = \text{Max}(C)$; |
| 2 else if $C$ has negative non-ground compound terms then $L = \text{Select}(C)$; |
| 3 else if $C$ has positive non-ground compound terms then $L = \text{Max}(C)$; |
| 4 else $L = \text{SelectTop}(C)$; |
| 5 if $C$ is ground then Apply BRes or TRes |
| 6 else if $C$ has negative non-ground compound terms then Apply BRes or TRes |
| 7 else Apply BRes or TRes |

The lexicographic path ordering is optional since one can use any admissible ordering that satisfies the precedence such that function symbols are larger than constant symbols, which are larger than predicate symbols.

In Algorithm 1, Max($C$) applies maximality principle with respect to $>_\text{lp}$ to $C$, and outputs the maximal literal. Select($C$) selects one of negative non-ground compound literals in $C$; and SelectTop($C$) is performed in two steps: i) All the negative literals in $C$ are selected so that Res is applicable on $C$, ii) Find top variables in $C$. In Step ii), if $C$ contains top variables, then all the negative literals in $C$ containing top variables are selected, else all the negative literals in $C$ remain selected. Select($C$) and SelectTop($C$) output selected literals.

Notice that SelectTop is not a standard selection, since, in a clause $C$, the selected literals may be not fixed. E.g., in Example 1, without modifying $Q$ and $C_2$, if we change $C_1$ to $A_3(f, x) \lor D(gx) \lor \neg G_2x$ and $C_3$ to $A_3(f, x) \lor \neg G_3x$, since $y$ and $z$ are top variables in $Q$, then all literals in $Q$ will be selected. The selected literals in $L$ are different from the selected ones in Example 1. In this paper, we justify SelectTop by proving that TRes is compatible with the framework of [3], since SelectTop enables only TRes, which is not a standard rule in the framework of [3].

We use T-Inf to denote a top-variable-based inference system containing rules: Deduct, Split, Del, Cond, and Fact, Res, BRes, TRes with the refinement T-Refine.

The following example gives a sample derivation.

**Example 4.** Consider an unsatisfiable set of loosely guarded clauses $C_1, \ldots, C_9$:

\[
\begin{align*}
C_1 &= \neg A_1xy \lor \neg A_2yz \lor \neg A_3xz \lor Bxyz \\
C_2 &= A_3(x, f) \lor \neg G_3x \\
C_3 &= A_2(f, x) \lor \neg G_2x \\
C_4 &= A_4(f, x, x) \lor D(gx) \lor \neg G_1x \\
C_5 &= \neg Bxxy \\
C_6 &= \neg Dx \\
C_7 &= G_1(fa) \\
C_8 &= G_3(fa) \\
C_9 &= G_2a
\end{align*}
\]

Suppose the precedence on which $>_\text{lp}$ is based is $f > g > a > B > A_1 > A_2 > A_3 > D > G_1 > G_2 > G_3$. We star the $>_\text{lp}$-maximal literals as $L^*$ and box the selected literals as $[L]$. 
Then $C_1 \ldots, C_9$ are presented as:

$$
C_1 = \neg A_1 xy \lor \neg A_2 yz \lor \neg A_3 zx \lor Bxyz
$$

$$
C_2 = A_1(x, f x) y \lor \neg G_1 x
$$

$$
C_3 = A_2(f x, f x)^* \lor \neg G_3 x
$$

$$
C_4 = A_1(f x, f x)^* \lor D(g x) \lor \neg G_1 x
$$

$$
C_5 = \neg Bxxy
$$

$$
C_6 = \neg Dx
$$

$$
C_7 = G_1(f a)^* \lor \neg G_3 x
$$

$$
C_8 = G_3(f a)^* \lor G_2 a^*
$$

One can use any clause to start the derivation, w.o.l.g. we start with $C_1$. For each new derived clause, we immediately apply T-Refine to determine the eligible literals in it.

1. Starting with $C_1$, since $C_1$ is a non-ground clause containing no non-ground compound term and multiple negative literals, one needs select all negative literals in $C_1$, and try to find top variables in $C_1$, to see whether we need to select negative literals in $C_1$ containing top variables, instead.

2. Applying Res on $C_1, C_2, C_3$ and $C_4$ produces an mgu $\{x / f x, y / f x, z / f x\}$ to substitute variables in $C_1$, then $x$ is the only top variable in $C_1$.

3. SelectTop selects literals $\neg A_1 xy$ and $\neg A_3 zx$ containing $x$. Then TRes is applied to $C_1, C_2$ and $C_4$, which derives:

$$
C_{10} = \neg A_2 xx \lor B(f(x), x, x)^* \lor D(g x) \lor \neg G_1 x \lor \neg G_3 x.
$$

4. Applying BRes to $C_{10}$ and $C_5$ derives $C_{11} = \neg A_2 xx \lor D(g x)^* \lor \neg G_1 x \lor \neg G_3 x$.

5. Applying BRes to $C_{11}$ and $C_5$ derives $C_{12} = \neg A_2 xx \lor \neg G_1 x \lor \neg G_3 x$.

6. Now we find top variables in $C_{12}$: Applying Res on $C_{12}, C_3, C_7$ and $C_8$ produce an mgu $\{x / f a\}$. Thus $x$ is the top variable in $C_{12}$. This means that all negative literals in $C_{12}$ remain being selected. Applying TRes on $C_{12}, C_3, C_7$ and $C_8$ derives $C_{13} = \neg G_3 x$.

7. Applying BRes to $C_{13}$ and $C_8$ derives $L$.

We now show that T-Inf is a sound and refutationally complete system.

**Theorem 1** ([3, 15, 41]). Let $N$ be a set of clauses that is saturated up to redundancy with respect to T-Inf. Then $N$ is unsatisfiable if and only if $N$ contains the empty clause.

In fact, if we take TRes as a form of ‘Partial ordered resolution with selection’ of Res, then we can show such ‘Partial ordered resolution with selection’ is compatible with the framework [3]. Base on Res, ‘Partial ordered resolution with selection’ is performed whenever Res is applicable: selected literals in the main premise of Res are changed to a subsequence of this main premise. Notice that the standard definition of selection function requires that once selected literals $L$ in a clause $C$ are set, $L$ are fixed whenever resolution is performed on $C$. Here we show that whenever Res is applicable, we can change selected literal in the main premise of Res so that ‘Partial ordered resolution with selection’ is applicable. We give proofs by using a variation of a sound rule in instantiation-based theorem proving [17].

**Lemma 1.** ‘Partial ordered resolution with selection’ is compatible with the framework [3].

**Proof.** In the proof, we take ‘Partial ordered resolution with selection’ as a combination of an instantiation-based inference rule Ins and ordered resolution with selection Res.

First we introduce Ins.

$$
\text{Ins: } \frac{B_1 \lor D_1 \ldots, B_k \lor D_k \lor \neg A_1 \lor \ldots \lor \neg A_n \lor D}{B_1 \lor \ldots \lor \neg A_1 \lor \ldots \lor \neg A_n \lor D}(\neg A_1 \lor \ldots \lor \neg A_n \lor D)\sigma
$$

where i) $B_1 \lor D_1, \ldots, B_k \lor D_k$ ($k \leq n$) and $\neg A_1 \lor \ldots \lor \neg A_n \lor D$ are the same as premises in Res. ii) $\sigma$ is an mgu for $A_1, \ldots, A_k$ and $B_1, \ldots, B_k$.

Ins is a sound: Given a clause $C$ and an arbitrary substitution $\sigma$, $C \models \sigma$. This implies that Ins is sound.

Ins adds a $\sigma$-instance of the main premise from Res to the first-order clausal set. If $\sigma'$ is the mgu defined in Res, then $\sigma$ is a more general mgu than $\sigma'$ such that there exists a substitution $\theta$ such that $\sigma' = \sigma \theta$. Selecting ($\neg A_1 \lor \ldots \lor \neg A_n \lor D$) in $C = (\neg A_1 \lor \ldots \lor \neg A_n \lor D)\sigma$ and applying Res on $B_1 \lor D_1, \ldots, B_k \lor D_k$ and $C$ derives the same resolvent as applying ‘Partial ordered resolution with selection’ on the premises in Ins. Since Ins is sound, and adding a sound rule Ins to the framework [3] do not alter refutational completeness and redundancy elimination in [3]. ‘Partial ordered resolution with selection’ is compatible with the framework. □

**5 Rewriting Query Clauses**

Now we are ready to give our query rewriting techniques. In this section, we are concerned with querying over guarded clauses, and in next section, we generalise our procedure to loosely guarded clauses.

We assume Cond and Split are immediately used on any clause whenever they are applicable. The requirement of using Cond is not critical for our results, but can give us a better view of how we rewrite query clauses.

First we define the notion of guard in first-order clauses. Let a negative flat literal $A$ and a subclause (or a literal) $D$ occur in a clause $C$. If $\var(D) \subseteq \var(A)$, we say $A$ guards $D$, and if $\var(C) = \var(A)$, we say $A$ is a guard of $C$. Hence, Condition 3 of the definition of the guarded clauses shows that $G$ is a guard of $C$.

**5.1 The Separation Rule Sep**

In our procedure, a separation rule and Split are used to remove literals containing isolated variables from query clauses. A variable $x$ is an isolated variable if $x$ is not chained. In a first-order clause $C$, a variable $x$ is a chained variable if there are two literals $A_1$ and $A_2$ in $C$ such that $\var(A_1) \notin \var(A_2), \var(A_2) \notin \var(A_1)$ and $x \in \var(A_1) \cap \var(A_2)$. E.g., in the query clause $\neg A_1 x \lor \neg A_2 y z \lor \neg A_3 y z$, $x$ and $z$ are isolated variables and $y$ is a chained variable.
Now we are ready to introduce the ‘Separation’ rule Sep.

Sep: \[
\frac{D_1 \lor A \lor D_2}{D_1 \lor A \lor q(x), \ \neg q(x) \lor D_2}
\]

where i) \(A\) contains both isolated variables and chained variables \(x\), which are shared with \(D_2\), ii) \(A\) guards \(D_1\), and ii) \(q\) is a fresh predicate symbol, called as an S-definer.

Sep is a replacement rule in which the premise is immediately replaced by the conclusions. The idea of using Sep is to separate out the subclause in the premise that contains isolated variables. For example, given a query clause \(Q = \neg A_1 x \lor \neg A_2 xy \lor \neg A_3 yz\), by introducing an S-definer \(q\), we can separate \(Q\) into \(C_1 = \neg A_1 x \lor \neg A_2 xy \lor q y\) and \(Q_1 = \neg q y \lor \neg A_3 yz\), so that the subclause \(\neg A_1 x \lor \neg A_2 xy\) that contains an isolated variable \(x\), is separated from \(Q\). Both \(C_1\) and \(Q_1\) are guarded clauses.

**Lemma 2.** On any first-order clause, Sep can be applied iteratively at most linearly often.

**Proof.** A first-order clause \(C = D_1 \lor A \lor D_2\) can be separated into \(C_1 = D_1 \lor A \lor q(x)\) and \(C_2 = q(x) \lor D_2\) only if \(A\) contains both isolated variables and chained variables \(x\) (w.r.t. \(C\)). In \(C_1\), because \(A\) guards \(C_1\), variables in \(D_1 \lor A \lor q(x)\) are isolated, thus we cannot apply Sep to \(C_1\).

Now we show that only literals in \(D_2\) can be separated from \(C_2\). Assume \(x\) and \(y\) are an isolated variable and a chained variable in \(x\) in \(C_2\), and \(x\) and \(y\) occur in literal \(L_x\) and \(L_y\) in \(D_2\), respectively. An isolated variable \(x\) occurs in two literals \(L_x\) and \(q\) implies that either \(\text{var}(q) \subseteq \text{var}(L_x)\) or \(\text{var}(L_x) \subseteq \text{var}(q)\). The latter implies \(A\) guards \(L_x\), thus impossible. It is the case that \(\text{var}(q) \subseteq \text{var}(L_x)\), all variables in \(q\) are isolated, which contradicts that \(y\) is a chained variable. This means that variables in \(q\) cannot contain both chained variables and isolated variables, hence, \(\neg q(x)\) cannot be separated out from \(C_2\).

We have shown that further application of Sep can only be applied to literals in \(D_2\). This implies that one can only apply Sep to a clause of length \(n\) at most \(n\) times. \(\square\)

Let us now discuss the possible logical consequences when applying Sep to query clauses. We say a query clause containing only chained variables is a chained-only query clause and a query clause containing only isolated variables is an isolated-only query clause. E.g., \(\neg A_1 x y \lor \neg A_2 y z \lor \neg A_3 x z\) is a chained-only query clause, and both \(\neg A_1 x y \lor \neg A_2 y z\) and \(\neg A_1 x y \lor \neg A_2 y z \lor \neg A_3 x z\) are isolated-only query clauses.

**Lemma 3.** An indecomposable isolated-only query clause is a guarded clause.

**Proof.** Let \(Q\) be an indecomposable isolated-only query clause. If there are two literal \(L_1\) and \(L_2\) in \(Q\) such that \(\text{var}(L_1) \nsubseteq \text{var}(L_2)\) and \(\text{var}(L_2) \nsubseteq \text{var}(L_1)\), then there are chained variables between \(L_1\) and \(L_2\). Because there are no such relations between literal pairs in \(Q\), for any pair of literals \(L_1\) and \(L_2\) in \(Q\), \(L_1\) guards \(L_2\) or vice versa. Then there exists a negative literal \(G\) in \(Q\) such that for any literal \(L\) in \(Q\), \(G\) guards \(L\). Therefore \(G\) guards \(Q\). Since \(Q\) is flat, negative and contains a negative literal \(\neg G\) that guards \(Q\), \(Q\) is a guarded clause. \(\square\)

Now we look at how Sep and Split rewrite a query clause:

**Lemma 4.** Exhaustively applying Sep and Split to a query clause \(Q\) transforms it into:

1. a set of guarded clauses, or
2. a set of guarded clauses and a chained-only query clause.

**Proof.** Assume a query clause \(Q = \neg D_1 \lor \neg A \lor C_2\) where i) \(A\) guards \(\neg D_1\), ii) \(A\) contains isolated variables and chained variables \(x\) that are shared with \(\neg D_2\). Then applying Sep to \(Q\) derives \(Q = \neg A \lor C_1 \lor C_2\). We say a guarded clause since \(A\) guards a flat clause \(C_1\), \(C_2\) is a query clause since it is negative and flat. This shows that each application of Sep to a query clause produces a guarded clause and a query clause. We cannot apply Sep to a guarded clause since the guard literal in it guards the clause. Then only the query clause \(C_2\) can be replaced by applying Sep.

Sep cannot be applied to a query clause \(Q\) if and only if \(Q\) does not contain a literal containing both isolated variables and chained variables. This implies that when Sep is not applicable to \(Q\), all variables in \(Q\) are either isolated or chained. According to Lemma 3, the former is a guarded clause, then condition 1 holds. Condition 2 holds for the latter. \(\square\)

Condition 1 in Lemma 4 means that in the final application of Sep to a query clause, an isolated-only query clause in the conclusion is a guarded clause.

Interestingly, we notice that given a query clause \(Q\), if we obtain a chained-only query clause \(Q_c\) from \(Q\) by exhaustively applying Sep and Split, the relations among chained variables \(x\) in \(Q_c\) is the same as the relations among \(x\) in \(Q\). In fact, we can regard Sep and Split as a syntactic implementation of a variation of GYO-reduction [40], where a cyclic query \(Q\) can be identified by recursively removing ‘ears’ in the hypergraph of \(Q\). In our context, ‘ears’ are literals containing isolated variables, and ‘ears’ are removed by Sep and Split. Acyclic and cyclic queries are defined using the notion of hypergraph, which can be found in [39]. By the property of GYO-reduction [40]:

**Lemma 5.** Exhaustively applying Sep and Split to a query clause \(Q\) transforms it into:

1. guarded clauses if \(Q\) is an acyclic query clause, or
2. guarded clauses and a chained-only query clause if \(Q\) is a cyclic query clause.

So far we have considered how Sep and Split rewrite query clauses. However, for chained-only query clauses such as \(\neg A_1 x y \lor \neg A_2 y z \lor \neg A_3 x z\), Sep and Split are not sufficient.

5.2 TRes and the Top Transformation T-Trans

Notice that in a chained-only query clause such as \(Q = \neg A_1 x y \lor \neg A_2 y z \lor \neg A_3 x z\), there exists a so-called ‘variable
cycle’ among \(x, y\) and \(z\) in \(Q\). To break this variable cycle, we employ resolution. However, naively applying unrefined resolution can easily cause variable depth increase in the derived clause, as shown in Example 1. Hence, we apply \(T\text{-Inf}\) to perform resolution on query clauses and guarded clauses to avoid variable depth increase in derived clauses:

**Lemma 6.** In an application of \(T\text{-Inf}\), no variable depth growth occurs in the resolvents of a chained-only query clause and a set of guarded clauses.

**Proof.** Since the eligible literals in an ordered resolution step on a chained-only query clause and a set of guarded clauses satisfy the conditions of a query pair clauses defined in [41], according to Theorem 1 in [41]: using \(T\text{Res}\), no variable depth growth occurs in the resolvents of a set of query pair clauses. Hence the lemma holds. \(\square\)

Example 5 shows how \(T\text{Res}\) breaks the cycle of \(x, y\) and \(z\) in the chained-only query clause \(Q\) while avoiding variable depth growth in the resolvent.

**Example 5.** Given a chained-only query clause \(Q\) and a set of guarded clauses \(C_1, \ldots, C_6:\)

\[
Q = \neg A_1 xy \lor \neg A_2 yz \lor \neg A_3 zx \lor \neg B_1 z u \lor \neg B_2 w u \lor \neg B_3 w z
\]

\(C_1 = A_1 (f x y, x) \lor D(g x y) \lor \neg G_1 x y\)

\(C_2 = A_2 (f x y, f x y) \lor \neg G_2 x y\)

\(C_3 = A_3 (x, f x y) \lor \neg G_3 x y\)

\(C_4 = B_1 (f x y, x) \lor \neg G_4 x y\)

\(C_5 = B_2 (f x y, f x y) \lor \neg G_5 x y\)

\(C_6 = B_3 (x, f x y) \lor \neg G_6 x y\)

using the mgu is \{\(y/f(f x y, y'), z/f(f x y, y'), u/f x y, w/f x y, x/f(f(f x y, y'), y')\}\) to substitute variables in \(Q\), we find that \(x\) is the top variable in \(Q\). Hence \(T\text{Res}\) is performed on \(Q, C_1\) and \(C_6\), and derives \(R = \neg G_1 x y \lor \neg G_3 x y \lor D(g x y) \lor \neg A_2 x x \lor \neg B_1 x u \lor \neg B_2 w u \lor \neg B_3 w x\).

Figure 2 illustrates the variable relations of the flat literals in \(Q\) and \(R\). We can see that the cycle among \(x, y\) and \(z\) in \(Q\) is broken. The new challenge in Example 5 is that \(R\) is neither a guarded clause nor a query clause. On such resolvents we use a structural transformation: we introduce a fresh predicate symbol \(p\) and use \(\neg p x y\) to replace the literals that are introduced to the query clause, so that \(R\) is transformed into: \(\neg G_1 x y \lor \neg G_3 x y \lor D(g x y) \lor p x y \lor \neg p x y \lor \neg A_2 x x \lor \neg B_1 x u \lor \neg B_2 w u \lor \neg B_3 w x\). The former is a guarded clause and the latter is a query clause.

![Figure 2. Variable relations (of flat literals) in Q and R](image)

The next definition makes precise how resolvents of \(T\text{Res}\) are transformed using structural transformation. There are two cases: i) top variables do not co-occur with each other in a literal of the main premise (the query clause) in \(T\text{Res}\), and ii) top variables co-occur with each other in a literal of the main premise in \(T\text{Res}\):

**Definition 6 (T-Trans, T-definer).** We assume the notions of \(T\text{Res}\). \(T\text{Trans}\) introduces a fresh predicate symbol \(q\), called a T-definer, in this manner:

1. If \(x\) is a top variable in the main premise \(Q\) in \(T\text{Res}\) such that i) \(x\) does not co-occur with any other top variable in any literal of \(Q\) and ii) \(x\) occurs in \(A_m, \ldots, A_m\) \((1 \leq m \leq m' \leq t)\), then we introduce a T-definer \(\neg q\) to define \((D_m \lor \ldots \lor D_m')\).

2. If \(x = x_1, \ldots, x_k\) is a set of top variables in the main premise \(Q\) in \(T\text{Res}\) such that i) for each \(x_i\) in \(x\), there exists a \(x_j\) in \(x\) such that \(x_i\) co-occurs with \(x_j\) in a literal of \(Q\) \((x_i \neq x_j)\), ii) variables in \(x_1, \ldots, x_k\) occur in \(A_m, \ldots, A_m'\) \((1 \leq m \leq m' \leq t)\), then we introduce a T-definer \(\neg q\) to define \((D_m \lor \ldots \lor D_m')\).

The idea behind \(T\text{Trans}\) is: considering the main premise is a chained-only query clause \(\neg A_1 \lor \ldots \lor \neg A_i \lor \ldots \lor \neg A_n\), the side premises are guarded clauses \(B_1 \lor D_1, \ldots, B_t \lor D_t\), and the mgu is \(\sigma\) as defined in \(T\text{Res}\). Take \(B_1 \lor D_1\) as an example, then removing a positive literal \(B_1\) from \(B_1 \lor D_1\) does not change the guardedness of \(D_1\). Knowing \(B_1, \ldots, B_t\) are guarded clauses, since non-ground compound terms in \(B_1, \ldots, B_t\) matches the same top variable, using the matching property (Lemma 3 in [41]) of \(T\text{Res}\), we can show that \((D_1 \lor \ldots \lor D_t)\) is a guarded clause. Hence, the resolvent \((D_1 \lor \ldots \lor D_t)\) \(\sigma\) is a guarded clause. Hence, \((D_1 \lor \ldots \lor D_t)\) \(\sigma\) and a query clause \((\neg A_1 \lor \ldots \lor \neg A_n)\) \(\sigma\) can be transformed into a guarded clause \((D_1 \lor \ldots \lor D_t)\) \(\sigma\) \(q\) and a query clause \((\neg A_1 \lor \ldots \lor \neg A_n)\) \(\neg q\). We give detailed proofs in the following lemma:

**Lemma 7.** Given a chained-only query clause \(Q\) and a set of guarded clauses \(C\), \(T\text{Trans}\) transforms the resolvents obtained by applying \(T\text{Res}\) to \(Q\) and \(C\) into a guarded clause and a query clause, of which the length is smaller than that of \(Q\).

**Proof.** Using \(T\text{Trans}\) where \(1 \leq m \leq m' \leq t, 1 \leq i' \leq n' \leq t\) with \(\sigma\) an mgu defined in \(T\text{Res}\), the resolvents in \(T\text{Res}\) are transformed into two types of clauses:

Type 1: \(Q_1 = \neg q_1 \lor \ldots \lor \neg q_{i'} \lor (\neg A_{i+1} \lor \ldots \lor \neg A_n)\) \(\sigma\),

Type 2: \(C_{i'} = q_{i'} \lor (D_m \lor \ldots \lor D_{m'})\) \(\sigma\).

First we show that \(Q_1\) is a query clause and the length of \(Q_1\) is smaller than that of the query clause in the main premise. According to the properties of top variable resolution, \(\sigma\) assigns variables in \(\neg A_{i+1} \lor \ldots \lor \neg A_n\) to only variables. Hence, \(Q_1\) is a query clause. Because the main premise \(Q\) in \(T\text{Res}\) is a query clause containing only chained variables, each top variable occurs in at least two different literals, let us say they are \(\neg A_1\) and \(\neg A_2\). Then in an application of \(T\text{Res}\), \(\neg A_1\) and \(\neg A_2\) are resolved upon, and \(D_1\) \(\sigma\) and \(D_2\) \(\sigma\) (\(\sigma\) is the
According to the matching in $\text{TRes}$ (variable $\text{Sep}$ an acyclic query clause, and satisfies the conditions of $\text{px}$ clause can show that $\text{px}$ clause. By using this proof for each of variable in $\text{TRes}$, the resolvents in $\text{TRes}$ is ground. This case is trivial. If $B_i$ is not ground, then $D_i$ is a guarded clause since removing a non-ground compound literal from a guarded clauses $C$ will not change the properties the of a guarded clause in Definition 4. We discuss two cases in $T\text{-Trans}$:

Case 1: Assume that $x$ is a top variable in $Q$ such that $x$ does not co-occur with any other top variables, and $x$ occurs in $A_{m_1}, \ldots, A_{m_l}$ and $\sigma$ is the mgu defined in $T\text{Res}$. Assume that $x$ matches $t_{m_1}, \ldots, t_{m_l}$ in $B_{m_1}, \ldots, B_{m_l}$, respectively, and the guard literals in $D_{m_1}, \ldots, D_{m_l}$ are $G_{m_1}, \ldots, G_{m_l}$, respectively. According to the matching in $T\text{Res}$, $x$ matches $t_{m_1}, \ldots, t_{m_l}$ implies that $t_{m_l} = \ldots = t_{m_1}$, hence $\text{var}(t_{m_l}) = \ldots = \text{var}(t_{m_1})$. Because $\text{var}(t_i) = \text{var}(G_i)(m_i \leq i \leq m_l)$, we get $\text{var}(G_m) = \ldots = \text{var}(G_1) = \text{var}(D_m)$. Since $\sigma$ assigns variables in $D_m \lor \ldots \lor D_{m_l}$ to only variables, $(D_m \lor \ldots \lor D_{m_l})$ is a guarded clause ($G_m, \ldots, G_{m_l}$ as the guards). Since $\text{var}(q_i) = \text{var}(G_m)$, $C_i = q_i \lor (D_m \lor \ldots \lor D_{m_l})$ is a guarded clause.

Case 2: Assume that $\bar{x} = x_1, \ldots, x_k$ is a set of top variables such that for each variable $x_i$ in $\bar{x}$, there is another variable $x_j$ such that $x_i$ co-occurs with $x_j$ in a literal $L$ of $Q$. Assume $x_i$ and $x_j$ is a pair in $x_1, \ldots, x_k$ such that $x_i$ and $x_j$ co-occur with each other, and $\sigma$ is the mgu defined in $T\text{Res}$. Assume $x_j$ matches $t_{l_1}, \ldots, t_{l_k}$ in $B_{l_1}, \ldots, B_{l_k}$, respectively where $m \leq i \leq m'$, and $x_j$ matches $t_{j_1}, \ldots, t_{j_k}$ in $B_{j_1}, \ldots, B_{j_k}$, respectively where $m \leq k \leq m'$. Proofs are similar to the one in Case 1 show that $(D_{l_1} \lor \ldots \lor D_{l_k})$ and $(D_{j_1} \lor \ldots \lor D_{j_k})$ are two guarded clauses. Assume in $L$, $t_{l_1}$ matches $x_i$ and $t_{j_1}$ matches $x_j$. Since $\text{var}(t_{l_1}) = \text{var}(t_{j_1})$, $\text{var}(t_{l_1}) = \text{var}(t_{j_1})$. We know that $\text{var}(D_{l_1}) = \text{var}(D_{l_1})$, $\text{var}(D_{l_1}) = \text{var}(D_{l_1})$, $\text{var}(D_{l_1}) = \text{var}(D_{l_1})$, and $\text{var}(D_{l_1}) = \text{var}(D_{l_1})$. Then $(D_{l_1} \lor \ldots \lor D_{l_k}) \lor (D_{j_1} \lor \ldots \lor D_{j_k})$ is a guarded clause. By using this proof for each of variable in $\bar{x}$, we can show that $(D_m \lor \ldots \lor D_{m_l})$ is a guarded clause. Since $\text{var}(q_i) = \text{var}(G_m)$ and $G_m$ is a guard in $(D_m \lor \ldots \lor D_{m_l})$, $C_i = q_i \lor (D_m \lor \ldots \lor D_{m_l})$ is a guarded clause.

Using $T\text{-Trans}$, $R$ in Example 5 produces a query clause $Q_1 = \neg px \lor \neg A_2 xx \lor B_1 x u \lor B_2 y w \lor \neg B_3 w x$ and a guarded clause $px y \lor G_1 x y \lor G_2 x y \lor D(g x y)$ with $g$ as a T-definer. We notice that the newly derived query clause $Q_1$ is actually an acyclic query clause, and satisfies the conditions of Sep since $\neg px y$ contains an isolated variable $y$ and a chained variable $x$. Hence using Sep, we can further separate $\neg px y \lor \neg A_2 xx$ from $Q_2$ by introducing an S-definer $q$, obtaining a guarded clause $Q_2 = \neg qx \lor \neg B_1 x u \lor \neg B_2 y w \lor \neg B_3 w x$, which is a chained-only query clause. One can apply $T\text{Res}$ to $Q_2$ and its respective side premises, to break the cycle in $Q_2$ and derives a resolvent that will be later transformed into a set of guarded clauses using $T\text{-Trans}$. Figure 3 shows the variable relations in $Q_1$ and $Q_2$ ($q$ is omitted since it is a unary literal). We can see that Sep removes isolated variables in $Q_1$.

5.3 The Query Rewriting Procedure $Q$-Proc

Noticing that all the ‘byproducts’ of Sep, $T\text{Res}$ and $T\text{-Trans}$ are guarded clauses, we realise that, given a query clause $Q$, these rules can produce a set of guarded clause from $Q$. In fact, we found that the given query clause will eventually be reduced to a guarded clause as well. The formal procedure $Q$-Proc to rewrite a query clause into a set of guarded clauses is shown in Algorithm 2.

In Algorithm 2, Sep() is a function that applies Sep to a query clause $Q$, outputting a guarded clause $C$, and either an isolated-only query clause (hence guarded, Lemma 3) or a chained-only query clause. We use $T\text{Res}()$ to denote a function that applies $T\text{Res}$ to a chained-only query clause and a set of guarded clauses, and outputs the resolvent $R$. We use $T\text{Trans}()$ to denote a function that applies $T\text{Trans}$ to $R$, deriving a guarded clause $C$ and a query clause $Q$. As always, in an application of $Q$-Proc, we assume $\text{Split}$ and $\text{Cond}$ are applied whenever applicable.

Now we can show the first main contribution of this paper that $Q$-Proc is able to rewrite query clauses into a set of guarded clauses within finitely many steps:

**Theorem 2.** $Q$-Proc rewrites query clauses into a set of guarded clauses in finitely bounded time.

**Algorithm 2: The query rewriting procedure $Q$-Proc**

**Input:** A query clause $Q$, guarded clauses $C$

**Output:** Guarded clauses $C'$

1. while $Q$ is not a guarded clause do
2. 
3. $Q, C = \text{Sep}(Q)$;
4. $C' = C \cup \{C\}$;
5. if $Q$ is a chained-only query clause then
6. $R = T\text{Res}(Q, C')$;
7. $Q, C = T\text{Trans}(R)$;
8. $C' = C' \cup \{C\}$;

Figure 3. Variable relations in $Q_1$ and $Q_2$.
Proof. Given an input query clause \( Q \), assume that after exhaustively applying \( \text{Sep} \) and \( \text{Split} \) to \( Q \), the output query clause is \( Q_1 \). The length of \( Q_1 \) is no more than that of \( Q \) since \( \text{Sep} \) and \( \text{Split} \) do not add literals to \( Q \). If \( Q_1 \) is an isolated-only query clause, then according to Lemma 3, \( Q_1 \) is a guarded clause. Thus \( Q\text{-Proc} \) terminates. If \( Q_1 \) is a chained-only query clause, then we apply \( \text{TRes} \) to \( Q_1 \) to produce the resolvents \( R \) and apply \( \text{T-Trans} \) \( R \), producing the query clause \( Q_2 \). Lemma 7 shows that the length of \( Q_2 \) is smaller than the that of \( Q_1 \), so that the length of new query clauses is decreasing. When the length of a query clause has decreased to 2, after applying \( \text{Sep} \) and \( \text{Split} \), the output clauses are guarded clauses (this can be justified that by applying \( \text{Sep} \) and \( \text{Split} \) to a length-two query clause, the conclusions are chained variable free, hence guarded). Therefore \( Q\text{-Proc} \) can rewrite a query clause into a set of guarded clauses within finitely many steps. \( \square \)

6 Querying GF and LGF

This section first shows that our query answering system \( Q\text{-Inf} \), given by the combination of the query rewriting procedure \( Q\text{-Proc} \) and the top-variable-based inference system \( T\text{-Inf} \) extended with \( \text{Sep} \) and \( \text{T-Trans} \), decides guarded clauses and loosely guarded clauses. Then we give the soundness and the completeness proof of \( Q\text{-Inf} \), and show that \( Q\text{-Inf} \) solves BCQ answering for GF and LGF.

Since \( T\text{-Inf} \) decides loosely guarded clauses \([41]\), which subsume guarded clauses. We show the same for \( Q\text{-Inf} \).

Theorem 3 ([41]). \( Q\text{-Inf} \) decides (loosely) guarded clauses.

Now we consider that given a query clause and a set of loosely guarded clause, can \( Q\text{-Inf} \) rewrite a query clause into a set of loosely guarded clause? The critical difference in \( \text{TRes} \) is that the side premises are loosely guarded rather than only guarded. Nevertheless, applying \( \text{TRes} \) to a query clause and loosely guarded clauses does not alter guard literals in the side premises, hence we can still apply the same techniques. We can use proofs in Lemma 7 to show:

Lemma 8. Given a chained-only query clause \( Q \) and a set of loosely guarded clauses \( C \), \( \text{T-Trans} \) transforms the resolvents obtained by applying \( \text{TRes} \) to \( Q \) and \( C \) into a set of loosely guarded clauses and a query clause, of which the length of \( Q_1 \) is smaller than that of \( Q \).

Example 6. Consider a chained-only query clause \( Q \) and a set of loosely guarded clauses \( C_1, \ldots, C_6 \):

\[
Q = \neg A_1 xy \lor \neg A_2 yz \lor \neg A_3 zx \lor \neg B_2 zu \lor \neg B_3 wv \lor \neg B_4 wz
\]

\[
C_1 = A_1 (fxyz, x) \lor D(gxyz) \lor \neg G_11 xy \lor \neg G_12 yz \lor \neg G_13 zx
\]

\[
C_2 = A_2 (fxyz, fxyz) \lor \neg G_21 xy \lor \neg G_22 yz \lor \neg G_23 zx
\]

\[
C_3 = A_3 (x, fxyz) \lor \neg G_31 xy \lor \neg G_32 yz \lor \neg G_33 zx
\]

\[
C_4 = B_1 (fxyz, x) \lor \neg G_41 xy \lor \neg G_42 yz \lor \neg G_43 zx
\]

\[
C_5 = B_2 (fxyz, fxyz) \lor \neg G_51 xy \lor \neg G_52 yz \lor \neg G_53 zx
\]

\[
C_6 = B_3 (x, fxyz) \lor \neg G_61 xy \lor \neg G_62 yz \lor \neg G_63 zx
\]

Like in Example 5, \( x \) in \( Q \) is still the top variable, hence \( \text{TRes} \) is performed on \( Q, C_1 \) and \( C_6 \), and derives \( R = \neg G_11 xy \lor \neg G_12 yz \lor \neg G_13 zx \lor \neg G_31 xy \lor \neg G_32 yz \lor \neg G_33 zx \lor D(gxyz) \lor \neg A_3 xx \lor \neg B_1 xu \lor \neg B_3 uw \lor \neg B_5 wx \). Then we can introduce a T-definer \( q \) such that \( R \) is transformed into \( \neg G_11 xy \lor \neg G_12 yz \lor \neg G_31 xy \lor \neg G_32 yz \lor \neg G_33 zx \lor D(gxyz) \lor qxyz \lor \neg qxyz \lor \neg A_3 xx \lor \neg B_1 xu \lor \neg B_3 uw \lor \neg B_5 wx \). The former is a loosely guarded clause and the latter is a query clause.

Now we can generalise \( Q\text{-Proc} \) by changing guarded clauses to loosely guarded clauses in Algorithm 2. We abusively still refer to the query rewriting procedure for loosely guarded clauses as \( Q\text{-Proc} \). Then we can show:

Theorem 4. \( Q\text{-Proc} \) rewrites query clauses into a set of loosely guarded clauses in finitely many steps.

Proof. Using Lemma 8 and Theorem 2 by changing guarded clauses to loosely guarded clauses. \( \square \)

Now we need to show \( Q\text{-Inf} \) is sound and refutational complete. Compared to \( T\text{-Inf} \), \( Q\text{-Inf} \) contains two new rules: \( \text{Sep} \) and \( \text{T-Trans} \). Here we show \( \text{Sep} \) preserves satisfiability equivalence, formally stated as:

Lemma 9. In an application of \( \text{Sep} \), the premise is satisfiable if and only if both the conclusions are satisfiable.

Proof. If: Apply resolution to the conclusions derives the corresponding premise.

Only if: Let the premise \( D_1 \lor A \lor D_2 \) be satisfiable. Suppose an interpretation \( I \models D_1 \lor A \lor D_2 \). Then for any ground substitutions \( \sigma, I \models (D_1 \lor A \lor D_2) \sigma \). We extend \( I \) into interpretation \( I' \) such that \( I' \models q(\overline{\sigma}) \) if and only if \( I \models D_2 \sigma \) is true for any ground substitution \( \sigma \) over the Herbrand universe. i) If \( I \models D_2 \sigma \), then \( I' \models q(\overline{\sigma}) \). Since \( I' \models q(\overline{\sigma}) \), \( I' \models (D_1 \lor A \lor q(\overline{\sigma})) \sigma \). I \models D_2 \sigma \) implies \( I' \models D_2 \sigma \), thus \( I' \models (\neg q(\overline{x}) \lor D_2) \sigma \). Since \( \sigma \) is an arbitrary ground substitution over Herbrand universe, \( I' \models D_1 \lor A \lor q(\overline{x}) \). Accordingly to \( I' \models (D_1 \lor A \lor q(\overline{x})) \sigma \) and \( I' \models D_2 \sigma \), \( I' \models (D_1 \lor A) \sigma \). Then \( I' \models (D_1 \lor A \lor q(\overline{x})) \sigma \), thus \( I' \models (D_1 \lor A \lor q(\overline{x})) \sigma \). Since \( \sigma \) is an arbitrary ground substitution over the Herbrand universe, \( I' \models D_1 \lor A \lor q(\overline{x}) \) and \( I' \models \neg q(\overline{x}) \lor D_2 \). \( \square \)

Lemma 9 shows that \( \text{Sep} \) is sound. Using similar proofs, we can show that any form of structural transformation is sound, formally stated as:

Lemma 10. Any structural transformation (including \( \text{T-Trans} \)) preserves satisfiability equivalence.

We know that adding a sound rule to an inference system does not alter the refutational completeness of the system. This implies that \( Q\text{-Inf} \) is sound and refutational complete.

Theorem 5. \( Q\text{-Inf} \) is a sound inference system.
Theorem 6. Let $N$ be a set of clauses that is saturated up to redundancy with respect to Q-Inf. Then $N$ is unsatisfiable if and only if $N$ contains the empty clause.

Combining the termination of Q-Inf on query clauses and (loosely) guarded clauses (Theorem 3, 4), the refutational completeness (Theorem 6) and the soundness (Theorem 5), we can conclude that:

Theorem 7. Q-Inf decides (loosely) guarded clauses and query clauses.

Now we give the second main contribution of this paper:

Theorem 8. The combination of Q-Inf and the clausal transformation G-Trans and B-Trans solves the problem of Boolean conjunctive query answering problem for the (loosely) guarded fragment.

7 Conclusion and Future Work

In this paper, we present the first, as far as we know, practical rewriting procedure Q-Proc that rewrites a query clause into a set of (loosely) guarded clauses, and the first, as far as we know, query answering system Q-Inf that solves BCQ answering for GF and LGF.

During the investigation of querying for GF and LGF, we found it is interesting that the same resolution-based techniques in automated reasoning are connected to techniques used in database research. Since the mainstream query answering procedure in database research uses a tableau-like chase approach [1], it would be interesting to see how resolution can equally transfer in the other direction, and would also be of interest to use resolution to solve query-related problems such as query optimisation and query answering.

Given a Boolean conjunctive query $q$ and (loosely) guarded formulas, our method Q-Inf produces a saturated set of clauses or $\bot$. The question arises: is it possible to pinpoint the clauses that are responsible for the entailment or the non-entailment of the query $q$, which would be related to view-based query answering [31] and query explanation problems.

Can we use the techniques in Q-Inf to develop decision procedures for satisfiability checking and query answering for other decidable fragments, such as fluted logic [33], the guarded negation fragment [7] and the clique guarded fragment [7]? As yet there are no practical decision procedure for the last two fragments, and no query answering procedure for all three of them.

Can we extend Q-Inf so that it can answer conjunctive queries rather than Boolean conjunctive queries, so that one can retrieve actual answers from the guarded fragment knowledge bases? This question has practical significance because it would allow us retrieving answers from fragments of the guarded fragment such as description logic $\mathcal{ALCH\Omega I}$ and guarded existential rules.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. Foundations of Databases: The Logical Level (1st ed.). Addison-Wesley Longman Publishing Co., Inc.
[2] Hajnal Andréka, István Németi, and Johan van Benthem. 1998. Modal Languages and Bounded Fragments of Predicate Logic. J. Philos. Logic 27, 3 (1998), 217–274.
[3] Leo Bachmair and Harald Ganzinger. 2001. Resolution Theorem Proving. In Handbok of Automated Reasoning, Alan Robinson and Andrei Voronkov (Eds.). Elsevier and MIT Press, 19–99.
[4] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. 1993. Superposition with Simplification as a Decision Procedure for the Monadic Class with Equality. In Proc. Computational Logic and Proof Theory, Springer, 83–96.
[5] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. 2011. On Rules with Existential Variables: Walking the Decidability Line. Artif. Int. 175, 9 (2011), 1620–1654.
[6] Vince Bárány, Georg Gottlob, and Martin Otto. 2010. Querying the Guarded Fragment. In Proc. LICS’10. IEEE Computer Society, 1–10.
[7] Vince Bárány, Balder ten Cate, and Luc Segoufin. 2015. Guarded Negation. J. ACM 62, 3 (2015), 22:1–22:26.
[8] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. 2009. Datalog+/-. A Unified Approach to Ontologies and Integrity Constraints. In Proc. ICDT’09. ACM, 14–30.
[9] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. 2007. Ontology-Based Database Access. In Proc. SEBD’07. SEBD, 324–331.
[10] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2007. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. J. Automat. Reasoning 39, 3 (2007), 385–429.
[11] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In Proc. 26th’77. ACM, 77–90.
[12] Hans de Nivelle and Maarten de Rijke. 2003. Deciding the Guarded Fragments by Resolution. J. Symb. Comput. 35, 1 (2003), 21–58.
[13] Nachum Dershowitz. 1982. Orderings for Term-rewriting Systems. Theoretical Comp. Sci. 17, 3 (1982), 279–301.
[14] C. Fermüller, A. Leitsch, T. Tammet, and N. Zamov. 1993. Resolution Methods for the Decision Problem. LNLI, Vol. 679. Springer.
[15] Harald Ganzinger and Hans de Nivelle. 1999. A Superposition Decision Procedure for the Guarded Fragment with Equality. In Proc. LICS’99. IEEE, 295–303.
[16] H. Ganzinger, U. Hustadt, C. Meyer, and R. A. Schmidt. 1998. A Resolution-Based Decision Procedure for Extensions of K4. In Proc. AiML’98. CSLI, 225–246.
[17] Harald Ganzinger and Konstantin Korovin. 2003. New Directions in Instantiation-Based Theorem Proving. In Proc. LICS’03 IEEE Computer Society, 55–64.
[18] Christophe Geissler and Kurt Konolige. 1986. A Resolution Method for Quantified Modal Logics of Knowledge and Belief. In Proc. TARK’86. Morgan Kaufmann, 309–324.
[19] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2002. Hypertree Decompositions and Tractable Queries. J. Computer and Sys. Sci. 64, 3 (2002), 579 – 627.
[20] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2003. Robbers, Marshals, and Guards: Game Theoretic and Logical Characterizations of Hypertree Width. J. Comp. and Syst. Sci. 66, 4 (2003), 775–808.
[21] Erich Grädel. 1999. Decision Procedures for Guarded Logics. In Proc. CADE’16. Springer, 31–51.
[22] Erich Grädel. 1999. On the Restraining Power of Guards. J. Symb. Logic 64, 4 (1999), 1719–1742.
[23] Colin Hirsch and Stephan Tobies. 2002. A Tableau Algorithm for the Clique Guarded Fragment. In Advances In Modal Logic, World
[24] Jan Hladik. 2002. Implementation and Optimisation of a Tableau Algorithm for the Guarded Fragment. In Proc. TABLEAUX’02. Springer, 145–159.

[25] Ian Hodkinson. 2002. Loosely Guarded Fragment of First-Order Logic Has the Finite Model Property. Studia Logica 70, 2 (2002), 205–240.

[26] Ullrich Hustadt. 1999. Resolution Based Decision Procedures for Subclasses of First-order Logic. Ph.D. Dissertation. Saarland Univ., Saarbrücken, Germany.

[27] Ullrich Hustadt and Renate A. Schmidt. 1997. On Evaluating Decision Procedures for Modal Logic. In Proc. IJCAI’97. Morgan Kaufmann, 202–207.

[28] U. Hustadt and R. A. Schmidt. 1999. Maslov’s Class K Revisited. In Proc. CADE’99 (LNAI). Springer, 172–186.

[29] Stanislav Kikot, Roman Kontchakov, and Michael Zakharyaschev. 2012. Conjunctive Query Answering with OWL 2 QL. In Proc. KR’12. AAAI, 275–285.

[30] Jose Mora, Riccardo Rosati, and Oscar Corcho. 2014. Kyrie2: Query Rewriting Under Extensional Constraints in EL′HOI. In Proc. ISWC’14. Springer, 568–583.

[31] Alan Nash, Luc Segoufin, and Victor Vianu. 2010. Views and Queries: Determinacy and Rewriting. ACM Trans. Database Syst. 35, 3 (2010), 41.

[32] Andreas Nonnengart and Christoph Weidenbach. 2001. Computing Small Clause Normal Forms. In Handbook of Automated Reasoning, Alan Robinson and Andrei Voronkov (Eds.). Elsevier and MIT Press, 335–367.

[33] Willard V Quine. 1960. Variables explained away. Proc. American Philosophical Society 104, 3 (1960), 343–347.

[34] Riccardo Rosati and Alessandro Almaitelli. 2010. Improving Query Answering over DL-Lite Ontologies. In Proc. KR’10. AAAI, 290–300.

[35] Renate A. Schmidt and Ullrich Hustadt. 2000. A Resolution Decision Procedure for Fluted Logic. In Proc. CADE’00. Springer, 433–448.

[36] Johan van Benthem. 1997. Dynamic bits and pieces. Research Report LP-97-01. Univ. Amsterdam.

[37] Moshe Y. Vardi. 1996. Why is Modal Logic So Robustly Decidable?. In Proc. DIMACS Workshop’96. DIMACS/AMS, 149–183.

[38] Moshe Y. Vardi. 2000. Constraint Satisfaction and Database Theory: A Tutorial. In Proc. PODS’00. ACM, 76–85.

[39] Mihalis Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In Proc. VLDB’81. VLDB Endowment, 82–94.

[40] C. T. Yu and M. Z. Ozsoyoglu. 1979. An Algorithm for Tree-query Membership of a Distributed Query. In Proc. COMPSAC’79. IEEE, 306–312.

[41] Sen Zheng and Renate A. Schmidt. 2020. Deciding the Loosely Guarded Fragment and Querying Its Horn Fragment Using Resolution. In Proc. AAAI’20. AAAI, Forthcoming.