Querying the Guarded Fragment via Resolution *

Sen Zheng and Renate A. Schmidt

Department of Computer Science, University of Manchester, UK

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 decision procedure to address this problem. In particular, we show that using a top-variable refinement, the separation rule and a form of dynamic renaming, one can rewrite Boolean conjunctive queries into a set of guarded clauses, so that querying the guarded fragment can be reduced to deciding the guarded fragment. As far as we know, this provides the first practical decision procedure for answering Boolean conjunctive queries over the guarded fragment.

1 Introduction

Answering queries over knowledge bases is at the heart of knowledge representation research. In this work, we are interested in the problem of answering Boolean conjunctive queries. A *Boolean conjunctive query* (BCQ) is a first-order formula of the form $q = \exists \overline{x} \varphi(\overline{x})$ where φ is a conjunction of atoms, in which only constants and variables are arguments. Given a Boolean conjunctive query q, a set of rules Σ and a database $\mathcal{D}$, our aim is to check whether $\Sigma \cup \mathcal{D} \models q$. Important problems in many research areas, such as query evaluation, query entailment [5] and query containment in database research [12], and constraint-satisfaction problem and homomorphism problems in general AI research [34] can be recast as BCQ answering problems.

In this work, we consider the case when the rules Σ are expressed in the guarded fragment [2]. Formulas in the guarded fragment (GF) are equlity-free first-order formulas without function symbols, in which the quantification is restricted to the form $\exists \overline{x}(G \wedge \varphi)$ such that the atom G contains all the free variables of φ. Satisfiability in many decidable propositional modal logics such as $\mathcal{K}$, $\mathcal{D}$, $\mathcal{S}3$, $\mathcal{S}4$ and $\mathcal{B}$ can be encoded as satisfiability of formulas in GF. GF inherits robust decidability, captured by the tree model property [33], from modal logic [21, 24], hence, there are intense investigation from a theoretical perspective for GF [20, 2, 21] and practical decision procedures have been developed for it [23, 13, 16, 37].

In ontology-mediated query answering systems [10], the description logic $\mathcal{ALCHOI}$ and its fragments [28, 11, 29, 30], and guarded existential rules [9] are commonly used ontological languages. A description axioms easily maps to guarded formulas in which the arities of predicate symbols and the number of variables are limited. Also, guarded existential rules are Horn guarded formulas. Querying GF is known to be 2ExpTime-complete [6], however, as yet there has been insufficient effort to develop practical querying procedures. In this work, we present a resolution decision procedure to solve BCQ answering problems in GF. Resolution provides a powerful method for developing practical decision procedures as has been shown in [13, 17, 16, 18, 25, 26, 4] for example.

One of the main challenges in this work is the handling of query formulas, since these formulas, e.g., $\exists xyz(Rxy \wedge Ryz)$, are beyond GF. By simply negating a BCQ, one can obtain

*We thank Uwe Waldmann and PAAR 2020 reviewers for the very useful comments. Sen Zheng's work is partially sponsored by the Great Britain-China Educational Trust.

a *query clause*: a clause containing only negative literals in which only variables and constants are arguments, such as $\neg Rxy \vee \neg Ryz$. One can take query clauses as (hyper-)graphs where variables are vertices and literals are edges. Then we use a separation rule **Sep** [31], which is also referred to as 'splitting through new predicate symbol' [27], and the splitting rule **Split** [3] to cut branches off query clauses. Each 'cut branch' follows the guardedness pattern, namely is a *guarded clause*. In general, we found that if a query clause Q is acyclic, one can rewrite Q into a set of guarded clauses by exhaustively applying separation and splitting to Q. That an acyclic BCQ can be equivalently rewritten as a guarded formula is also reflected in other works [19, 7]. If a query clause is cyclic, after cutting all branches, one can obtain a new query clause Q that only consists of variable cycles, i.e., each variable in Q connects two distinct literals that share some overlapping variables. We use top variable resolution **TRes** to handle such query clauses, so that by resolving multiple literals in Q, the variable cycles are broken. Then we use a dynamic renaming technique **T-Trans**, to transform a **TRes**-resolvent into a query clause and a set of guarded clauses. We show that only finitely many definers are introduced by **Sep** and **T-Trans**.

Top variable resolution **TRes** is inspired by the 'MAXVAR' technique in deciding the loosely guarded fragment [13, 16], which later adjusted in [37] to solve BCQs answering problem over the Horn loosely guarded fragment. Interestingly, we discovered that separation and splitting in query rewriting behaves like GYO-reduction represented in [36], where cyclic queries q [35] are identified by recursively removing 'ears' in the hypergraph of q. A similar query rewriting procedure is 'squid decomposition' [8], aiming to rewrite BCQs over Datalog$^{+/-}$ using the chase approach [1]. In a squid decomposition, a query is regarded as a squid-like graph in which branches are 'tentacles' and variable cycles are 'heads'. Squid decomposition finds ground atoms that are complementary in the squid head, then ground unit resolution is used to eliminate the heads. Our approach first uses **Sep** and **Split** to cut all 'tentacles', and then uses **TRes** to break cycles in 'heads'. Hence, grounding is not necessary. By appropriately applying separation, splitting, top variable resolution and a form of dynamic renaming, query clauses can be effectively rewritten into a set of guarded clauses or be shown that no further inference on these query clauses are necessary.

Having a set of guarded clauses, another task is building an inference system to reason with these clauses. Existing inference systems for GF are either based on tableau (see [23, 22]) or resolution (see [13, 16, 37]). Our aim is to develop an inference system in line with the framework in [3], as it provides a powerful system unifying many different resolution refinement that exist in different forms of standard resolution, hyper-resolution and selection-based resolution. We develop our system as a variation of [16, 37], which are the only existing systems that decide GF, so that we can take advantage of simplification rules and notions of redundancy elimination. In particular, our inference system can be combined with the rewriting procedure, giving us as a query answering system for answering BCQs for GF.

2 Preliminaries

Let **C**, **F**, **P** denote pairwise disjoint discrete sets of *constant symbols* c, *function symbols* f and *predicate symbols* P, respectively. A *term* is either a variable or a constant or an expression $f(t_1, \ldots, t_n)$ where f is a n-ary function symbol and $t_1, ..., t_n$ 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 P is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms. A *literal* is an atom A (a *positive literal*) or a negated atom $\neg A$ (a *negative literal*). The terms $t_1, \ldots, t_n$ in literal $L = P(t_1, \ldots, t_n)$ are the *arguments* of L. A *first-order clause* is a multiset of literals, presenting a disjunction of literals. An *expression* can be a term, an atom, a literal, or a clause. A *compound-term* literal (clause) is a literal (clause) that contains at least one compound term argument.

A *substitution* is a mapping defined on variables, where variables denoting terms are mapped to terms. By $E\sigma$ we denote the result of applying the substitution σ 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 σ such that $E'\sigma = E\sigma$. A substitution σ 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 θ of s and t, there is a substitution ρ such that $\sigma\rho = \theta$. $\sigma\rho$ denotes the composition of σ and ρ as mappings. A *simultaneous most general unifier* σ is an 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 $1 \leq i \leq n$. As common, we use the term mgu to denote the notion of simultaneous mgu.

We use $\mathrm{dep}(t)$ to denote the depth of a term t, formally defined as: if t is a variable or a constant, then $\mathrm{dep}(t) = 0$; and if t is a compound term $f(u_1, \ldots, u_n)$, then $\mathrm{dep}(t) = 1 + max(\{\mathrm{dep}(u_i) \mid 1 \leq i \leq n\})$. In a first-order clause C, the *length* of C means the number of literals occurring in C, denoted as $\mathrm{len}(C)$, and the *depth* of C means the deepest term depth in C, denoted as $\mathrm{dep}(C)$. Let $\overline{x}$, $\overline{A}$, $\mathcal{A}$, $\mathcal{C}$ denote a sequence of variables, a sequence of atoms, a set of atoms and a set of clauses, respectively. Let $\mathrm{var}(t)$, $\mathrm{var}(C)$ and $\mathrm{var}(\overline{A_n})$ be sets of variables in a term t, a clause C and a sequence of atoms $\overline{A_n}$, respectively.

The rule set Σ denotes a set of first-order formulas and the database $\mathcal{D}$ denotes a set of ground atoms. A *Boolean conjunctive query* (BCQ) q is a first-order formula of the form $\exists\overline{x}\varphi(\overline{x})$ where φ is a conjunction of atoms, in which arguments are only constants and variables. Thus we can answer a Boolean conjunctive query $\Sigma \cup D \models q$ by checking whether $\Sigma \cup D \cup \neg q \models \bot$. In this work, we particularly focus on the case when Σ is expressed in GF without function symbols and equality.

3 From Logic Fragments to Clausal Sets

In this section, we provide the formal definitions of GF and define a structural transformation so that guarded formulas and BCQs can be converted into suitable sets of clauses.

Definition 1 (Guarded Fragment)**.** *Without equality and function symbols, the* guarded fragment (GF) *is a class of first-order formulas, inductively defined 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. *Let F belong to GF and G be an atom. Then $\forall\overline{x}(G \rightarrow F)$ and $\exists\overline{x}(G \wedge F)$ belong to GF if all free variables of F are among variables of G. G is referred to* guard.

Clausal Transformation. We now introduce the clausal transformation for GF and BCQs. We use **Q-Trans** to denote our clausal transformation, which is a variation of the structural transformation used in [13, 16, 37]. We explicitly assume that all free variables are existentially quantified, and formulas are transformed into prenex normal form before Skolemisation. Due to the page limit, we refer readers to [15] for detailed notions of clausal transformation techniques.

If an input formula is a BCQ, then we simply negate the BCQ to obtain a query clause. Using **Q-Trans**, a guarded formula F can be transformed into a set of clauses as follows:

1. Add existential quantifiers for all free variables in F and transform F into negation normal form, obtaining the formula F_{nnf}.
2. Apply the structural transformation: introduce fresh predicate symbols $d^i_\forall$ for universally quantified subformulas, obtaining F_{str}.
3. Transform F_{str} into prenex normal form and apply Skolemisation, obtaining F_{sko}.
4. Drop all universal quantifiers and transform F_{sko} into conjunctive normal form, obtaining a set of guarded clauses.

A literal L is *flat* if each argument in L is either a constant or a variable. A literal L is *simple* [16] if each argument in L is either a variable or a constant or a compound term $f(u_1, \ldots, u_n)$ where each u_i is a variable or a constant. A clause C is called *simple* (*flat*) if all literals in C are simple (flat). A clause C is *covering* if each compound term t in C satisfies that $\mathrm{var}(t) = \mathrm{var}(C)$.

Definition 2. *A* query clause *is a flat first-order clause containing only negative literals.*

Definition 3. *A* guarded clause *C is a simple and covering first-order clause satisfying the following conditions:*

1. *C is either ground, or*
2. *C contains a negative flat literal $\neg G$ satisfying that* $\mathrm{var}(C) = \mathrm{var}(G)$. *$G$ is referred to as* guard.

4 Top Variable Inference System

In this section, we present the top variable based inference system from [37], inspired by [13], which is enhanced with the splitting rule. The system is defined in the spirit of [3] and provides a decision procedure for the loosely guarded fragment and querying the Horn loosely guarded fragment [37]. The loosely guarded fragment [32] strictly subsumes GF by allowing multiple guards that enjoy variable co-occurrence property. Based on the system in [37], we build a system for querying the whole of GF.

Let $\succ$ be a strict ordering, called a *precedence*, on the symbols in $\mathbf{C}$, $\mathbf{F}$ and $\mathbf{P}$. An ordering $\succ$ on expressions is *liftable* if $E_1 \succ E_2$ implies $E_1\sigma \succ E_2\sigma$ for all expressions E_1, E_2 and all substitutions σ. An ordering $\succ$ on literals is *admissible*, if i) it is well-founded and total on ground literals, and liftable, ii) $\neg A \succ A$ for all ground atoms A, iii) if $B \succ A$, then $B \succ \neg A$ for all ground atoms A and B. A ground literal L is *$\succ$-maximal with respect to a ground clause C* if for any L' in C, $L \succeq L'$, and L is *strictly $\succ$-maximal with respect to* C if for any L' in C, $L \succ L'$. 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 σ such that $L\sigma$ is (strictly) maximal with respect to $C\sigma$, that is, for all L' in C, $L\sigma \succeq L'\sigma$ ($L\sigma \succ L'\sigma$). A *selection function* $\mathrm{Select}(C)$ selects a possibly empty set of occurrences of negative literals in a clause C with no other 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 $\mathrm{Select}(C)$ and L is a $\succ$-maximal literal with respect to C, or L is selected by $\mathrm{Select}(C)$.

As a default setting, all premises in resolution rules are variable-disjoint. The top variable based inference system contains following rules:

Deduct: $\dfrac{N}{N \cup \{C\}}$ if C is a conclusion of either **Res**, or **TRes**, or **Fact**, derived from clauses in N.

Split: $$\frac{N \cup \{C \vee D\}}{N \cup \{C\} \mid N \cup \{D\}}$$ if C and D are non-empty and variable-disjoint.

Fact: $$\frac{C \vee A_1 \vee A_2}{(C \vee A_1)\sigma}$$ if i) no literal is selected in C, ii) A_1 is $\succ$-maximal with respect to C. σ is an mgu of A_1 and A_2.

Res: $$\frac{B \vee D_1 \qquad \neg A \vee D}{(D_1 \vee D)\sigma}$$ if i) either A is selected, or nothing is selected in $\neg A \vee D$ and $\neg A$ is the maximal literal, ii) B is strictly $\succ$-maximal with respect to D_1. σ is an mgu of A and B.

TRes: $$\frac{B_1 \vee D_1 \quad \ldots \quad B_m \vee D_m \quad \ldots \quad B_n \vee D_n \qquad \neg A_1 \vee \ldots \vee \neg A_m \vee \ldots \vee \neg A_n \vee D}{(D_1 \vee \ldots \vee D_m \vee \neg A_{m+1} \vee \ldots \vee \neg A_n \vee D)\sigma}$$

if i) there exists an mgu σ' such that $B_i\sigma' = A_i\sigma'$ for each i such that $1 \leq i \leq n$, making $\neg A_1 \vee \ldots \vee \neg A_m$ *top-variable literals* and being selected, and D is positive, iv) no literal is selected in $D_1, \ldots, D_n$ and $B_1, \ldots, B_n$ are strictly $\succ$-maximal with respect to $D_1, \ldots, D_n$, respectively. σ is an mgu such that $B_i\sigma = A_i\sigma$ for all i such that $1 \leq i \leq m$.

The *top-variable literals* are computed using ComputeTop$(C_1, \ldots, C_n, C)$ in three steps:

1. Without producing or adding the resolvent, compute an mgu σ' among $C_1 = B_1 \vee D_1, \ldots, C_n = B_n \vee D_n$ and $C = \neg A_1 \vee \ldots \vee \neg A_n \vee D$ such that $B_i\sigma' = A_i\sigma'$ for each i satisfying that $1 \leq i \leq n$.
2. Compute the variable order $>_v$ and $=_v$ over variables in $\neg A_1 \vee \ldots \vee \neg A_n$: $x >_v y$ if $\mathrm{dep}(x\sigma') > \mathrm{dep}(y\sigma')$ and $x =_v y$ if $\mathrm{dep}(x\sigma') = \mathrm{dep}(y\sigma')$.
3. Based on $>_v$ and $=_v$, identify the maximal variables in $\neg A_1 \vee \ldots \vee \neg A_n$, which we call the *top variables*. The *top-variable literals* for an application of **TRes** to C are literals in C containing at least one top variable.

We use *T-Refine* to denote the following resolution refinement: i) a lexicographic path ordering $\succ_{lpo}$ [14] based on a precedence that any function symbol is larger than constant symbols, and any constant symbol is larger than predicate symbols, ii) selection functions and iii) Algorithm 1, which determines applications of $\succ_{lpo}$ and selection functions on clauses.

Algorithm 1 computes for a given clause C, the eligible literals in it. Eligible literals are either the (strictly) $\succ_{lpo}$-maximal literals in C, denoted as Max(C); or selected literals in C,

Algorithm 1: Computing eligible literals in a clause C

Input: A query clause or a guarded clause C
Output: Eligible literals in C

```
if C is ground then
    return Max(C);               ▷ Negative or positive premise in Res or TRes
else if C has negative compound terms then
    return Select(C);            ▷ Negative premises in Res
else if C has positive compound terms then
    return Max(C);               ▷ Positive premises in Res or TRes
else if C is a guarded clause then
    return SelectG(C);           ▷ Negative premises in Res
else
    return SelectT(C);           ▷ Negative premises in TRes
```

denoted as: Select(C), SelectG(C) and SelectT(C). Select(C) selects one of negative compound literals in C, SelectG(C) selects one of guards in C, and SelectT(C) is described in Algorithm 2 below.

Algorithm 2: Computing eligible literals using SelectT

```
Select all negative literals in C, denoted as L_all;
Find side premises C_1, ..., C_n of C ;            ▷ Satisfying Condition i) in TRes
if C_1, ..., C_n are in the given clausal set then
    L_top = ComputeTop(C_1, ..., C_n, C) ; ▷ Computing top-variable literals in C
    return L_top ;                                  ▷ TRes is applicable
else return L_all ; ▷ Select L_all if there are no adequate side premises for C
```

We use *T-Inf* to denote a top variable based inference system containing the rules: **Deduct**, **Split**, **Fact**, **Res**, **TRes** using the refinement *T-Refine*.

Theorem 1 ([3, 37, 16])**.** *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 an empty clause.*

5 Rewriting Query Clauses

We use the separation and splitting rules to 'cut branches' in query clauses. A clause is *indecomposable* if it cannot be partitioned into two non-empty variable-disjoint subclauses. Using **Split**, any decomposable query clause can be reduced to a set of indecomposable query clauses. Hence from now on, we assume all query clauses are indecomposable query clauses.

Given a query clause Q, we use the notion of *surface literal* to divide variables in Q into two kinds of variables, i.e., *chained-variables* and *isolated variables*. We say L is a *surface literal* in a query clause Q if for any L' in Q that is distinct from L, $\mathrm{var}(L) \not\subset \mathrm{var}(L')$. Let surface literals in a query clause Q be $L_1, \dots, L_n$ where $n \geq 1$. Then the *chained variables* in Q are variables among $\bigcup\limits_{i,j \in n} \mathrm{var}(L_i) \cap \mathrm{var}(L_j)$ whenever $\mathrm{var}(L_i) \neq \mathrm{var}(L_j)$, i.e., variables that link distinct surface literals containing different sets of variables, and *isolated variables* are the other non-chained variables. Now we can present the separation rule:

$$\textbf{Sep}: \quad \frac{N \cup \{C \vee A \vee D\}}{N \cup \{C \vee A \vee d_s(\overline{x}), \neg d_s(\overline{x}) \vee D\}}$$

if i) $\overline{x} = \mathrm{var}(A) \cap \mathrm{var}(D)$, ii) $\mathrm{var}(C) \subseteq \mathrm{var}(A)$, iii) A contains isolated variables, iv) d_s is a fresh predicate symbol.

Sep is a replacement rule in which $C \vee A \vee D$ is immediately replaced by $C \vee A \vee d_s(\overline{x})$ and $\neg d_s(\overline{x}) \vee D$.

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., $\neg A(x_1, x_2) \vee \neg B(x_2, x_3) \vee \neg C(x_3, x_4) \vee \neg D(x_4, x_1)$ is a chained-only query clause where x_1, x_2, x_3 and x_4 are all chained variables, whereas $\neg A(x_1, x_2, x_3) \vee \neg B(x_2, x_3)$ is an isolated-only query clause where x_1, x_2 and x_3 are all isolated variables. According to the definition of chained variables, if a query clause Q contains no chained variables, then either Q contains only one surface literal, or all surface literals in Q share the same variables. Therefore

Lemma 1. *An indecomposable isolated-only query clause is a guarded clause.*

Now we look at how **Sep** rewrites indecomposable query clauses.

Lemma 2. *Exhaustively applying **Sep** to an indecomposable query clause Q transforms Q 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** rewrites query clauses. However, **Sep** itself is not sufficient to handle chained-only query clauses such as $\neg A_1xy \vee \neg A_2yz \vee \neg A_3xz$, where there exists a so-called 'variable cycle' among x, y and z. We employ the **TRes** rule to break such variable cycles while avoiding term depth increase in derived clauses.

Example 1. *Given a chained-only query clause Q and a set of guarded clauses $C_1, \ldots, C_6$:*

$Q = \neg A_1xy \vee \neg A_2yz \vee \neg A_3zx \vee \neg B_1zu \vee \neg B_2uw \vee \neg B_3wz$
$C_1 = A_1(fxy, x) \vee D(gxy) \vee \neg G_1xy$ $\quad C_2 = A_2(fxy, fxy) \vee \neg G_2xy$ $\quad C_3 = A_3(x, fxy) \vee \neg G_3xy$
$C_4 = B_1(fxy, x) \vee \neg G_4xy$ $\quad C_5 = B_2(fxy, fxy) \vee \neg G_5xy$ $\quad C_6 = B_3(x, fxy) \vee \neg G_6xy$

ComputeTop$(Q, C_1, \ldots, C_6)$ *computes the mgu $\sigma' = \{x/f(f(f(x_1, y_1), y'), y^*), y/f(f(x_1, y_1), y'), u/f(x_1, y_1), z/f(f(x_1, y_1), y'), w/f(x_1, y_1)\}$ among Q and $C_1, \ldots, C_6$. Hence x is the only top variable in Q, so that **TRes** is performed on Q, C_1 and C_3, deriving $R = \neg G_1xy \vee \neg G_3xy \vee D(gxy) \vee \neg A_2xx \vee \neg B_1xu \vee \neg B_2uw \vee \neg B_3wx$.*

The first two figures in Figure 1 illustrate the variable relations of the flat literals in query clause Q and in **TRes**-resolvent R of Example 1. A cycle among x, y and z in Q is broken by **TRes**. The new challenge in Example 1 is that R is neither a guarded clause nor a query clause. On such resolvents we use the following structural transformation: we introduce fresh predicate symbols d_t, and use $\neg d_txy$ to replace the literals that are introduced to the query clause, so that R is transformed into: $\neg G_1xy \vee \neg G_3xy \vee D(gxy) \vee d_txy$ and $\neg d_txy \vee \neg A_2xx \vee \neg B_1xu \vee \neg B_2uw \vee \neg B_3wx$. The former is a guarded clause and the latter is a query clause.

Definition 4. *Let **TRes** derive the resolvent $(\neg A_{m+1} \vee \ldots \vee \neg A_n \vee D_1 \vee \ldots \vee D_m \vee D)\sigma$ using guarded clauses $A_1 \vee D_1, \ldots, A_n \vee D_n$ as the side premises, a chained-only query clause $Q = \neg A_1 \vee \ldots \vee \neg A_n$ as the main premise and a substitution σ such that $B_i\sigma = A_i\sigma$ for all i such that $1 \leq i \leq m$ as an mgu. Then **T-Trans** introduces fresh predicate symbols d_t, called* T-definer, *to transform R into a set of clauses, in this manner: Let $X_1, \ldots, X_t$ be top variables in Q. Then we partition $X_1, \ldots, X_t$ into sets $\mathcal{S}$ such that i) each pair of sets contain no common variable, and ii) each pair of variables in a set co-occurs in a literal of Q. Then for each set in $\mathcal{S}$ containing variables $\mathcal{X}$, we introduce a* T-definer *for $\mathcal{D}\sigma$ if $\mathcal{X}$ occur in $\mathcal{A}$.*

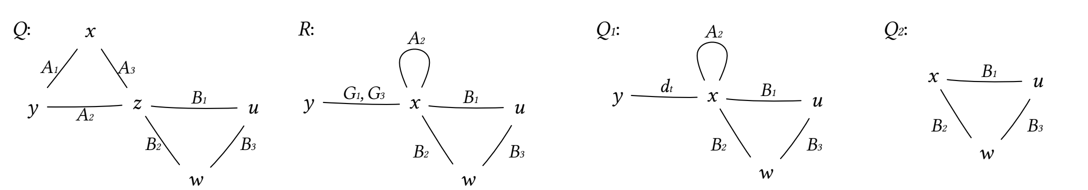

Figure 1: Variable relations of flat literals in Q, R, Q_1 and Q_2. From Q to R, **TRes** breaks the variable cycle among x, y and z in Q. From R to Q_1, **T-Trans** transforms R into a query clause Q_1. From Q_1 to Q_2, **Sep** cut off branches containing A_2 and d_t, from Q_1.

Lemma 3. *Let Q be a chained-only query clause and $\mathcal{C}$ be a set of guarded clauses, **T-Trans** transforms **TRes**-resolvents of Q and $\mathcal{C}$ into a set of guarded clauses and a query clause, of which the length is smaller than that of Q.*

Using **T-Trans**, R in Example 1 produces a query clause $Q_1 = \neg d_t xy \vee \neg A_2 xx \vee \neg B_1 xu \vee \neg B_2 uw \vee \neg B_3 wx$ and a guarded clause $d_t xy \vee \neg G_1 xy \vee \neg G_3 xy \vee D(gxy)$ with a T-definer d_t. The newly derived query clause Q_1 has branches, hence one can use **Sep** to cut the branch $\neg d_t xy \vee \neg A_2 xx$ from Q_1 by introducing an S-definer d_s, obtaining a guarded clause $d_s x \vee \neg d_t xy \vee \neg A_2 xx$ and a query clause $Q_2 = \neg d_s x \vee \neg B_1 xu \vee \neg B_2 uw \vee \neg B_3 wx$, which is a chained-only query clause. Then one can break the cycle in Q_2 by **TRes** and derives a resolvent that can be later renamed into guarded clauses using **T-Trans**. The last two figures in Figure 1 show the variable relations in Q_1 and Q_2 (the unary $\neg d_s x$ is omitted). We can see how **Sep** has cut off Q_1's branches.

Noticing that all the 'byproducts' of **Sep**, **TRes** and **T-Trans** are guarded clauses, we realise that, given a query clause Q, these rules only produce guarded clauses. In fact, we found that the given query clause will eventually be reduced to either a guarded clause or chained-only query clauses that no inferences can be performed on. Algorithm 3 formally describe such a query rewriting procedure, namely **Q-Rewrite**. Sep(Q) 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 2) or a chained-only query clause. TRes($Q, \mathcal{C}'$) denotes a function that applies **TRes** to a chained-only query clause Q and a set of guarded clauses $\mathcal{C}'$, and outputs the resolvent R. T-Trans(R) is a function that applies **T-Trans** to R, deriving a set of guarded clause C and a query clause Q.

Algorithm 3: Query rewriting procedure **Q-Rewrite**

Input: A query clause Q, a set of guarded clauses $\mathcal{C}$
Output: A set of guarded clauses $\mathcal{C}'$ and possibly chained-only query clauses

while *Q is not a guarded clause* **do**
    $Q, C = \text{Sep}(Q)$; ▷ `Apply Sep to the given query clause` Q
    $\mathcal{C}' = \mathcal{C} \cup \{C\}$;
    if *Q is a chained-only query clause* **then**
        if ***TRes** is applicable on Q* **then**
            $R = \text{TRes}(Q, \mathcal{C}')$; ▷ `Apply TRes to chained-only query clauses` Q
            $Q, C = \text{T-Trans}(R)$; ▷ `Apply T-Trans to the TRes resolvents`
            $\mathcal{C}' = \mathcal{C}' \cup \{C\}$;
        else
            return $Q, \mathcal{C}'$; ▷ `No rule can be performed on` Q
return $\mathcal{C}'$

6 Querying the Guarded Fragment

Since it is known that *T-Inf* decides guarded clauses [16, 13], we consider the new rules **Sep** and **T-Trans**. We show the new rules preserve satisfiability equivalence, therefore

Lemma 4. *In any application of **Sep** and **T-Trans**, the premise is satisfiable if and only if the conclusions are satisfiable.*

Lemma 5. ***Sep*** *and* ***T-Trans*** *only introduce a finitely bounded number of definers.*

We can show that *T-Inf* combined with **Q-Rewrite** is sound and refutationally complete.

Theorem 2. *Let N be a set of clauses that is saturated up to redundancy with respect to* T-Inf *and* ***Q-Rewrite****. Then, N is unsatisfiable if and only if N contains an empty clause.*

We can conclude that:

Theorem 3. T-Inf *and* ***Q-Rewrite*** *decides guarded clauses and query clauses. Hence together with the clausal transformation* ***Q-Trans****,* T-Inf *and* ***Q-Rewrite*** *solve the problem of Boolean conjunctive query answering for the guarded fragment.*

7 Conclusion and Future Work

In this paper, we present, as far as we know, the first practical rewriting procedure **Q-Rewrite** that rewrites a query clause into a set of clauses that can be decide by *T-Inf*, and as far as we know, the first query answering system that solves BCQ answering for the guarded fragment.

During the investigation of querying for the guarded fragment, we found it interesting that the same resolution-based techniques in automated reasoning are connected to techniques found in the database literature. Since the mainstream query answering procedure in database research uses a tableau-like chase approach [1], it would be interesting to see how a resolution-based approach performs in practice. We will implement the proposed procedure and conduct empirical evaluations as future works.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] Hajnal Andréka, István Németi, and Johan van Benthem. Modal languages and bounded fragments of predicate logic. *J. Philos. Logic*, 27(3):217–274, 1998.

[3] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.

[4] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In *Proc. Computational Logic and Proof Theory*, volume 713 of *LNCS*, pages 83–96. Springer, 1993.

[5] Jean-François Baget, Michel Leclére, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Int.*, 175(9):1620–1654, 2011.

[6] Vince Bárány, Georg Gottlob, and Martin Otto. Querying the guarded fragment. In *Proc. LICS'10*, pages 1–10. IEEE Computer Society, 2010.

[7] Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. *J. ACM*, 62(3):22:1–22:26, 2015.

[8] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Int. Res.*, 48(1):115–174, 2013.

[9] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog+/-: A unified approach to ontologies and integrity constraints. In *Proc. ICDT'09*, pages 14–30. ACM, 2009.

[10] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. Ontology-based database access. In *Proc. SEBD'07*, pages 324–331. SEBD, 2007.

[11] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Automat. Reasoning*, 39(3):385–429, 2007.

[12] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. SToC'77*, pages 77–90. ACM, 1977.

[13] Hans de Nivelle and Maarten de Rijke. Deciding the guarded fragments by resolution. *J. Symb. Comput.*, 35(1):21–58, 2003.

[14] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Comp. Sci.*, 17(3):279–301, 1982.

[15] Christian G. Fermüller, Alexander Leitsch, Ullrich Hustadt, and Tanel Tammet. Resolution decision procedures. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1791–1849. Elsevier and MIT Press, 2001.

[16] Harald Ganzinger and Hans de Nivelle. A superposition decision procedure for the guarded fragment with equality. In *Proc. LICS'99*, pages 295–303. IEEE, 1999.

[17] Harald Ganzinger, Ullrich Hustadt, Christoph Meyer, and Renate A. Schmidt. A resolution-based decision procedure for extensions of K4. In *Proc. AiML'98*, pages 225–246. CSLI, 1998.

[18] Christophe Geissler and Kurt Konolige. A resolution method for quantified modal logics of knowledge and belief. In *Proc. TARK'86*, pages 309–324. Morgan Kaufmann, 1986.

[19] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width. *J. Comp. and Syst. Sci.*, 66(4):775–808, 2003.

[20] Erich Grädel. Decision procedures for guarded logics. In *Proc. CADE'16*, volume 1632 of *LNCS*, pages 31–51. Springer, 1999.

[21] Erich Grädel. On the restraining power of guards. *J. Symb. Logic*, 64(4):1719–1742, 1999.

[22] Colin Hirsch and Stephan Tobies. A tableau algorithm for the clique guarded fragment. In *Advances In Modal Logic*, pages 257–277. World Scientific, 2002.

[23] Jan Hladik. Implementation and optimisation of a tableau algorithm for the guarded fragment. In *Proc. TABLEAUX'02*, volume 2381 of *LNCS*, pages 145–159. Springer, 2002.

[24] Ian Hodkinson. Loosely guarded fragment of first-order logic has the finite model property. *Studia Logica*, 70(2):205–240, 2002.

[25] Ullrich Hustadt. *Resolution Based Decision Procedures for Subclasses of First-order Logic*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 1999.

[26] Ullrich Hustadt and Renate A. Schmidt. On evaluating decision procedures for modal logic. In *Proc. IJCAI'97*, pages 202–207. Morgan Kaufmann, 1997.

[27] Yevgeny Kazakov. *Saturation-Based Decision Procedures for Extensions of the Guarded Fragment*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2006.

[28] Stanislav Kikot, Roman Kontchakov, and Michael Zakharyaschev. Conjunctive query answering with OWL 2 QL. In *Proc. KR'12*, pages 275–285. AAAI, 2012.

[29] Jose Mora, Riccardo Rosati, and Oscar Corcho. Kyrie2: Query rewriting under extensional constraints in $\mathcal{ELHOI}$. In *Proc. ISWC'14*, volume 8796 of *LNCS*, pages 568–583. Springer, 2014.

[30] Riccardo Rosati and Alessandro Almatelli. Improving query answering over DL-Lite ontologies. In *Proc. KR'10*, pages 290–300. AAAI, 2010.

[31] Renate A. Schmidt and Ullrich Hustadt. A resolution decision procedure for fluted logic. In *Proc. CADE'00*, volume 1831 of *LNCS*, pages 433–448. Springer, 2000.

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

[33] Moshe Y. Vardi. Why is modal logic so robustly decidable? In *Proc. DIMACS Workshop'96*, pages 149–183. DIMACS/AMS, 1996.

[34] Moshe Y. Vardi. Constraint satisfaction and database theory: A tutorial. In *Proc. PODS'00*, pages 76–85. ACM, 2000.

[35] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proc. VLDB'81*, pages 82–94. VLDB Endowment, 1981.

[36] Clement Yu and Meral Ozsoyoglu. An algorithm for tree-query membership of a distributed query. In *Proc. COMPSAC'79*, pages 306–312. IEEE, 1979.

[37] Sen Zheng and Renate A. Schmidt. Deciding the loosely guarded fragment and querying its Horn fragment using resolution. In *Proc. AAAI'20*, pages 3080–3087. AAAI, 2020.