Querying the Guarded Fragment via Resolution (Extended Abstract)

Sen Zheng, 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 the top-variable inference system, the separation rule and a form of dynamic renaming, one can derive a saturated set of query clauses and guarded clauses. As far as we know, this provides the first practical decision procedure for answering Boolean conjunctive queries over the guarded fragment.

Keywords

Resolution Decision Procedure, Guarded Fragment, Boolean Conjunctive Query Answering

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 $\Sigma \cup \mathcal{D} \models q$. Important problems in many research areas, such as query evaluation, query entailment [1] and query containment in database research [2], and constraint-satisfaction problem and homomorphism problems in general AI research [3] can be recast as a BCQ answering problem.

In this work, we consider the case when the rules Σ are expressed in the guarded fragment [4]. Formulae in the guarded fragment (GF) are equlity-free first-order formulae without function symbols, in which the quantification is restricted to the forms of $\forall \overline{x}(G \rightarrow \varphi)$ and $\exists \overline{x}(G \wedge \varphi)$ such that the atom G contains all the free variables of φ. Satisfiability in many decidable propositional modal logics such as $\mathcal{K}$, $\mathcal{D}$, $\mathcal{S}5$ and $\mathcal{T}$ can be encoded as satisfiability of formulae in GF (using the standard translation to first-order logic [5, Chapter 2]). GF inherits robust decidability, captured by the tree model property [6], from modal logic [7, 8], hence, there are intense investigation from a theoretical perspective for GF [9, 4, 7] and practical decision procedures have been developed for it [10, 11, 12, 13].

In ontology-mediated query answering systems [14], the description logic $\mathcal{ALCHOI}$ and its fragments [15, 16, 17, 18], and guarded existential rules [19] are commonly used ontological languages. A description logic axiom can easily map to a guarded formula in which the arities

PAAR 2020: Seventh Workshop on Practical Aspects of Automated Reasoning, June 29–30, 2020, Paris, France (virtual)
0000-0003-4415-811X (S. Zheng); 0000-0002-6673-3333 (R. A. Schmidt)

of predicate symbols and the number of variables are limited. Also, guarded existential rules can be seen as Horn guarded formulae. Querying GF is known to be 2ExpTime-complete [20], 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 [11, 21, 12, 22, 23, 24, 25] for example.

One of the main challenges in this work is the handling of BCQs, since these formulae, e.g., $\exists xyz(Rxy \land Ryz)$, are beyond GF. By simply negating a BCQ, one can obtain a *query clause*: a clause containing only negative literals in which only variables and constants are arguments, such as $\neg Rxy \lor \neg Ryz$. One can take query clauses as (hyper-)graphs where variables are vertices and literals are edges. Then we use a separation rule **Sep** [26] (which is also referred to as 'binary splitting rule with naming' [27]) and the splitting rule **Split** [28] 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 transform 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 [29, 30]. If a query clause is cyclic, after cutting off its branches, one can obtain a query clause Q that only consists of variable cycles, i.e., each variable in Q connects two distinct literals that share non-inclusive variable sets. 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 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 [11, 12], which later adjusted in [13] 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 [31], where cyclic queries [32] are identified by recursively removing 'ears' in the hypergraph of the given cyclic queries. A similar query rewriting procedure is 'squid decomposition' [33], aiming to rewrite BCQs over Datalog$^{+/-}$ using the chase approach [34]. 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 uses ground unit resolution 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.

Another task is building an inference system to reason with guarded clauses. Existing inference systems for GF are either based on tableau (see [10, 35]) or resolution (see [11, 12, 13]). Our aim is to develop an inference system in line with the framework in [28], 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 [12, 13], which are the only existing systems that decide GF, so that we can take advantage of simplification rules and notions of redundancy elimination.

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 $f(t_1, \dots, t_n)$ 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 $P(t_1, \dots, t_n)$, where P is an n-ary predicate symbol and $t_1, \dots, t_n$ are terms. A *literal* is an atom A (a *positive literal*) or a negated atom $\neg A$ (a *negative literal*). The terms $t_1, \dots, t_n$ in literal $L = P(t_1, \dots, t_n)$ 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.

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 $f(u_1, \dots, u_n)$, 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 $\overline{x}$, $\mathcal{X}$, $\mathcal{A}$, $\mathcal{C}$ denote a sequence of variables, a set of variables, a set of atoms and a set of clauses, respectively. Let $\mathrm{var}(t)$, $\mathrm{var}(E)$ be a set of variables in a an expression E.

The rule set Σ denotes a set of first-order formulae 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 the satisfiability of $\Sigma \cup D \cup \neg q$. 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 formulae 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 formulae, 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 *an atom. Then* $\forall\overline{x}(G \to F)$ *and* $\exists\overline{x}(G \land F)$ *belong to* GF *if all free variables of* F *are among variables of* G. G *is referred to as* 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 [11, 12, 13]. We explicitly assume that all free variables are existentially quantified. Due to the page limit, we refer readers to [36] 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 subformulae, obtaining F_{str}.
3. Transform F_{str} into prenex normal form and then apply Skolemisation, obtaining F_{sko}.
4. Drop all universal quantifiers and transform F_{sko} into conjunctive normal form, obtaining a set of guarded clauses.

We use the following notions to formally define *query clauses* and *guarded clauses*. A literal is *flat* if each argument in it is either a constant or a variable. A literal is *simple* [12] if each argument in it 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 $\mathrm{var}(t) = \mathrm{var}(C)$.

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* $\mathrm{var}(C) = \mathrm{var}(G)$. G *is referred to as* guard.

4. Top Variable Inference System

In this section, we present the top variable based inference system from [13], inspired by [11], which is enhanced with the splitting rule. The system is defined in the spirit of [28] and provides a decision procedure for the loosely guarded fragment and querying the Horn loosely guarded fragment [13]. The loosely guarded fragment [37] strictly subsumes GF by allowing multiple guards that enjoy variable co-occurrence property. Based on the system in [13], we build a system for querying the whole of GF.

Let $\succ$ be a strict ordering, called a *precedence*, on the symbols in **C**, **F** and **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 (*strictly*) *$\succ$-maximal with respect to a ground clause* C if for any L' in C, $L \succeq L'$ ($L \succ L'$). 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 $\mathrm{Select}(C)$.

The top variable based inference system *T-Inf* containing the rules: **Deduct**, **Split**, **Fact**, **Res**, **TRes** using the refinement *T-Refine*, given as follows.

Deduct: $$\frac{N}{N \cup \{C\}}$$ 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 no literal is selected in C, and A_1 is maximal w.r.t. $C \vee A_1 \vee A_2$. 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 w.r.t. D_1. The premises are variable disjoint and σ is an mgu of A and B.

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

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 w.r.t. $D_1, \ldots, D_n$, respectively. The premises are variable disjoint and 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 $\mathrm{ComputeTop}(C_1, \ldots, C_n, C)$ 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 $\succ_{lpo}$ [38] 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 computes the eligible literals based on $\succ_{lpo}$ and selection functions. Given a clause C, eligible literals in it are either the (strictly) $\succ_{lpo}$-maximal literals, denoted as $\mathrm{Max}(C)$; or selected literals, denoted as: $\mathrm{Select}(C)$, $\mathrm{SelectG}(C)$ and $\mathrm{SelectT}(C)$. $\mathrm{Select}(C)$ selects one of negative compound literals in C, $\mathrm{SelectG}(C)$ selects one of guards in C, and $\mathrm{SelectT}(C)$ is described in Lines 11–16 of **Algorithm 1**.

Theorem 1 ([28, 13, 12]). *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.*

Algorithm 1: Computing eligible literals in a clause C

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

Function $\mathrm{SelectT}(C)$:
    Select all negative literals $\mathcal{L}$ in $C$
    Find positive premises $C_1, \ldots, C_n$ of $C$
    if $C_1, \ldots, C_n$ exist then return $\mathrm{ComputeTop}(C_1, \ldots, C_n, C)$
    else return $\mathcal{L}$
```

5. Handling Query Clauses

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

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, $\mathrm{var}(L) \not\subseteq \mathrm{var}(L')$. Let surface literals in a query clause Q be $L_1, \ldots, L_n$ where $n \geq 1$. Then the *chained variables* in Q are variables among $\bigcup_{i,j \in n} \mathrm{var}(L_i) \cap \mathrm{var}(L_j)$ whenever $\mathrm{var}(L_i) \neq \mathrm{var}(L_j)$, i.e., variables that link distinct surface literals containing non-inclusive variable sets, 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) A is negative surface literal containing both chained variables $\overline{x}$ and isolated variables $\overline{y}$, ii) $\overline{x} \subseteq \mathrm{var}(D)$ and $\overline{y} \not\subseteq \mathrm{var}(D)$, iii) $\mathrm{var}(C) \subseteq \mathrm{var}(A)$, iv) d_s is a *definer.*

Sep is a replacement rule in which the premise $C \vee A \vee D$ is immediately replaced by its conclusions $C \vee A \vee d_s(\overline{x})$ 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** handles indecomposable query clauses.

Lemma 2. *Exhaustively applying **Sep** to an indecomposable 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** handles 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 $C_1, \ldots, C_6$:*

$$Q = \neg A_1xy \vee \neg A_2yz \vee \neg A_3zx \vee \neg B_1zu \vee \neg B_2uw \vee \neg B_3wz$$

$$C_1 = A_1(fxy, x) \vee D(gxy) \vee \neg G_1xy \qquad C_2 = A_2(fxy, fxy) \vee \neg G_2xy$$

$$C_3 = A_3(x, fxy) \vee \neg G_3xy \qquad C_4 = B_1(fxy, x) \vee \neg G_4xy$$

$$C_5 = B_2(fxy, fxy) \vee \neg G_5xy \qquad C_6 = B_3(x, fxy) \vee \neg G_6xy$$

$\mathrm{ComputeTop}(Q, C_1, \ldots, C_6)$ *computes the mgu* $\{x/f(f(f(x_1, y_1), y'), y^*), y/f(f(x_1, y_1), y'), u/f(x_1, y_1), z/f(f(x_1, y_1), y'), w/f(x_1, y_1)\}$ *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 this example 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

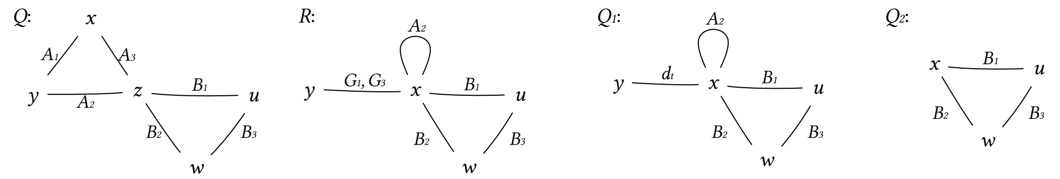

Figure 1: Variable relations of flat literals in Q, R, Q_1 and Q_2. From Q to R, **TRes** breaks the variable cycle among x, y and z in Q. From R to Q_1, **T-Trans** transforms R into a query clause Q_1. From Q_1 to Q_2, **Sep** cut off branches containing A_2 and d_t, from Q_1.

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.

We apply denote such structural transformation as **T-Trans** and apply it as the following manner: 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 positive premises, a chained-only query clause $Q = \neg A_1 \vee \ldots \vee \neg A_n$ as the negative 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 to transform R into a set of clauses, in the following manner: Let $x_1, \ldots, x_t$ be top variables in Q. Then we partition $x_1, \ldots, x_t$ into sets $\mathcal{S}_1, \ldots, \mathcal{S}_s$ 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 $\mathcal{S}_1, \ldots, \mathcal{S}_s$ containing variables $\mathcal{X}$, if $\mathcal{X}$ occur in $\mathcal{A}$, we introduce a definer d_t for $\mathcal{D}\sigma$.

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 $\mathcal{C}$ (if **TRes** is applicable) 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 $Q_1 = \neg d_txy \vee \neg A_2xx \vee \neg B_1xu \vee \neg B_2uw \vee \neg B_3wx$ and a guarded clause $d_txy \vee \neg G_1xy \vee \neg G_3xy \vee D(gxy)$ with a T-definer d_t. The newly derived query clause Q_1 has branches, hence one can use **Sep** to cut the branch $\neg d_txy \vee \neg A_2xx$ from Q_1 by introducing a definer d_s, obtaining a guarded clause $d_sx \vee \neg d_txy \vee \neg A_2xx$ and a query clause $Q_2 = \neg d_sx \vee \neg B_1xu \vee \neg B_2uw \vee \neg B_3wx$, 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

Algorithm 2: Saturation procedure of query clauses and guarded clauses **Q-Saturate**

```
Input: A query clause $Q$, a set of guarded clauses $\mathcal{S}$
1  $\mathcal{S}_Q, Q \leftarrow \mathrm{SepSplit}(Q)$
2  $\mathcal{S} \leftarrow \mathcal{S}_Q \cup \mathcal{S}$
3  if $Q$ is a chained-only query clause then
4      return SaturateCOQC($\mathcal{S}, Q$)
5  else return SaturateGC($\mathcal{S} \cup Q$);
6
7  Function SaturateCOQC($\mathcal{S}, Q$):
8      if $Q$ is a guarded clause then return SaturateGC($\mathcal{S} \cup Q$);
9      else if i) TRes is not applicable to $Q$ and SaturateGC($\mathcal{S}$) or ii) all inferences
         between $Q$ and SaturateGC($\mathcal{S}$) are redundant then
10         return SaturateGC($\mathcal{S}$) $\cup Q$
11     else
12         $R \leftarrow \mathrm{TRes}(C_1, \ldots, C_n, Q)$ ;          ▷ $C_1, \ldots, C_n \in$ SaturateGC($\mathcal{S}$)
13         $\mathcal{S}_R, Q' \leftarrow$ T-Trans($R$)
14         $\mathcal{S}', Q' \leftarrow$ SepSplit($Q'$)
15         $\mathcal{S} \leftarrow \mathcal{S}_R \cup \mathcal{S}' \cup \mathcal{S}_{sat}$
16         return SaturateCOQC($\mathcal{S}, Q$) $\cup$ SaturateCOQC($\mathcal{S}, Q'$)
```

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** 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 from Q. In fact, we found that the given query clause will eventually be reduced to either a guarded clause or chained-only query clauses. **Algorithm 2** formally describe the procedure to saturate query clauses and guarded clause, namely **Q-Saturate**. $\mathrm{SepSplit}(Q)$ is a function that recursively applies **Sep** and **Split** to a query clause Q, outputting guarded clauses $\mathcal{S}_Q$, and either an isolated-only query clause (a guarded clause, **Lemma 1**) or a chained-only query clause Q. $\mathrm{SaturateCOQC}(\mathcal{S}, Q)$ is a function that saturates guarded clauses $\mathcal{S}$ and a chained-only query clause Q. $\mathrm{SaturateGC}(\mathcal{S})$ is a function that uses *T-Inf* system to saturate guarded clauses $\mathcal{S}$. $\mathrm{TRes}(C_1, \ldots, C_n, Q)$ denotes a function that applies **TRes** to guarded clauses $C_1, \ldots, C_n$ and a chained-only query clause Q, and outputs the resolvent R. $\text{T-Trans}(R)$ is a function that applies **T-Trans** to the **TRes**-resolvent R, deriving guarded clause $\mathcal{S}_R$ and a query clause Q'.

6. Querying the Guarded Fragment

Since it is known that *T-Inf* decides guarded clauses [12, 11], we consider the new rules **Sep** and **T-Trans**. The new rules preserve satisfiability equivalence:

Lemma 4. *In any application of **Sep** and **T-Trans**, the premise is satisfiable if and only if its 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-Saturate** 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-Saturate**. Then, N is unsatisfiable if and only if N contains an empty clause.*

Theorem 3. ***Q-Saturate** decides guarded clauses and query clauses. Together with the clausal transformation **Q-Trans**, **Q-Saturate** solves BCQ answering for GF.*

7. Conclusion and Future Work

In this paper, we present, as far as we know, the first practical query answering procedure **Q-Saturate** that solves BCQ answering for GF. During the investigation, 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 [34], 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.

Acknowledgments

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.

References

[1] J.-F. Baget, M. Leclére, M.-L. Mugnier, E. Salvat, On rules with existential variables: Walking the decidability line, Artif. Int. 175 (2011) 1620–1654.

[2] A. K. Chandra, P. M. Merlin, Optimal implementation of conjunctive queries in relational data bases, in: Proc. SToC'77, ACM, 1977, pp. 77–90.

[3] M. Y. Vardi, Constraint satisfaction and database theory: A tutorial, in: Proc. PODS'00, ACM, 2000, pp. 76–85.

[4] H. Andréka, I. Németi, J. van Benthem, Modal languages and bounded fragments of predicate logic, J. Philos. Logic 27 (1998) 217–274.

[5] P. Blackburn, M. d. Rijke, Y. Venema, Modal Logic, Cambridge Tracts in Theoretical Computer Science, Cambridge Univ. Press, 2001.

[6] M. Y. Vardi, Why is modal logic so robustly decidable?, in: Proc. DIMACS Workshop'96, DIMACS/AMS, 1996, pp. 149–183.

[7] E. Grädel, On the restraining power of guards, J. Symb. Logic 64 (1999) 1719–1742.

[8] I. Hodkinson, Loosely guarded fragment of first-order logic has the finite model property, Studia Logica 70 (2002) 205–240.

[9] E. Grädel, Decision procedures for guarded logics, in: Proc. CADE'16, volume 1632 of *LNCS*, Springer, 1999, pp. 31–51.

[10] J. Hladik, Implementation and optimisation of a tableau algorithm for the guarded fragment, in: Proc. TABLEAUX'02, volume 2381 of *LNCS*, Springer, 2002, pp. 145–159.

[11] H. de Nivelle, M. de Rijke, Deciding the guarded fragments by resolution, J. Symb. Comput. 35 (2003) 21–58.

[12] H. Ganzinger, H. de Nivelle, A superposition decision procedure for the guarded fragment with equality, in: Proc. LICS'99, IEEE, 1999, pp. 295–303.

[13] S. Zheng, R. A. Schmidt, Deciding the loosely guarded fragment and querying its Horn fragment using resolution, in: Proc. AAAI'20, AAAI, 2020, pp. 3080–3087.

[14] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, Ontology-based database access, in: Proc. SEBD'07, SEBD, 2007, pp. 324–331.

[15] S. Kikot, R. Kontchakov, M. Zakharyaschev, Conjunctive query answering with OWL 2 QL, in: Proc. KR'12, AAAI, 2012, pp. 275–285.

[16] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The DL-Lite family, J. Automat. Reasoning 39 (2007) 385–429.

[17] J. Mora, R. Rosati, O. Corcho, Kyrie2: Query rewriting under extensional constraints in $\mathcal{ELHOI}$, in: Proc. ISWC'14, volume 8796 of *LNCS*, Springer, 2014, pp. 568–583.

[18] R. Rosati, A. Almatelli, Improving query answering over DL-Lite ontologies, in: Proc. KR'10, AAAI, 2010, pp. 290–300.

[19] A. Calì, G. Gottlob, T. Lukasiewicz, Datalog+/-: A unified approach to ontologies and integrity constraints, in: Proc. ICDT'09, ACM, 2009, pp. 14–30.

[20] V. Bárány, G. Gottlob, M. Otto, Querying the guarded fragment, in: Proc. LICS'10, IEEE, 2010, pp. 1–10.

[21] H. Ganzinger, U. Hustadt, C. Meyer, R. A. Schmidt, A resolution-based decision procedure for extensions of K4, in: Proc. AiML'98, CSLI, 1998, pp. 225–246.

[22] C. Geissler, K. Konolige, A resolution method for quantified modal logics of knowledge and belief, in: Proc. TARK'86, Morgan Kaufmann, 1986, pp. 309–324.

[23] U. Hustadt, Resolution Based Decision Procedures for Subclasses of First-order Logic, Ph.D. thesis, Univ. Saarlandes, Saarbrücken, Germany, 1999.

[24] U. Hustadt, R. A. Schmidt, On evaluating decision procedures for modal logic, in: Proc. IJCAI'97, Morgan Kaufmann, 1997, pp. 202–207.

[25] L. Bachmair, H. Ganzinger, U. Waldmann, Superposition with simplification as a decision procedure for the monadic class with equality, in: In Proc. KGC'93, volume 713 of *LNCS*, Springer, 1993, pp. 83–96.

[26] R. A. Schmidt, U. Hustadt, A resolution decision procedure for fluted logic, in: Proc. CADE'00, volume 1831 of *LNCS*, Springer, 2000, pp. 433–448.

[27] A. Riazanov, A. Voronkov, Splitting Without Backtracking, Research Report CSPP-10, Univ. Manchester, 2001.

[28] L. Bachmair, H. Ganzinger, Resolution theorem proving, in: A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning, Elsevier and MIT Press, 2001, pp. 19–99.

[29] G. Gottlob, N. Leone, F. Scarcello, Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width, J. Comp. and Syst. Sci. 66 (2003) 775–808.

[30] V. Bárány, B. ten Cate, L. Segoufin, Guarded negation, J. ACM 62 (2015) 22:1–22:26.

[31] C. Yu, M. Ozsoyoglu, An algorithm for tree-query membership of a distributed query, in: Proc. COMPSAC'79, IEEE, 1979, pp. 306–312.

[32] M. Yannakakis, Algorithms for acyclic database schemes, in: Proc. VLDB'81, VLDB Endowment, 1981, pp. 82–94.

[33] A. Calì, G. Gottlob, M. Kifer, Taming the infinite chase: Query answering under expressive relational constraints, J. Artif. Int. Res. 48 (2013) 115–174.

[34] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases: The Logical Level, Addison-Wesley Longman Publishing Co., Inc., 1995.

[35] C. Hirsch, S. Tobies, A tableau algorithm for the clique guarded fragment, in: Proc. AiML'00, World Scientific, 2000, pp. 257–277.

[36] M. Baaz, U. Egly, A. Leitsch, J. Goubault-Larrecq, D. A. Plaisted, Normal form transformations, in: J. A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning (in 2 volumes), Elsevier and MIT Press, 2001, pp. 273–333.

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

[38] N. Dershowitz, Orderings for term-rewriting systems, Theoretical Comp. Sci. 17 (1982) 279–301.