SATURATION-BASED QUERY ANSWERING AND REWRITING PROCEDURES FOR GUARDED FIRST-ORDER FRAGMENTS

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Science and Engineering

2021

Sen Zheng

Department of Computer Science

Contents

Word Count: 57295

List of Figures

List of Algorithms

List of Rules

Abstract

This thesis presents the first practical Boolean conjunctive query answering and the first saturation-based Boolean conjunctive query rewriting procedures for the guarded fragment and its extensions: the loosely guarded, the clique guarded, the guarded negation and the clique guarded negation fragments. All these fragments are robustly decidable, hence they are exceptionally qualified candidates as logical formalisms. The problems of answering Boolean conjunctive queries in all of these fragments are also decidable, nonetheless it is open whether there exist practical decision procedures for these problems. We close this gap by developing a theoretical framework for practical query answering procedures for all of these fragments, presenting new techniques, new inference systems and new procedures. In particular we devise a partial selection-based resolution rule, based on which we establish new, elegant and powerful saturation-based systems, named the top-variable inference systems. We formally prove the system are sound and refutationally complete for first-order clausal logic (with equality). Using these systems, we devise the first resolution-based decision procedure for the clique guarded fragment, and the first practical decision procedures for the unary negation, the guarded negation and the clique guarded negation fragments.

Another significant contribution is the presentation of saturation-based rewriting approaches, allowing a new perspective to the topic of query rewriting through the use of powerful automated deduction techniques. Our rewriting procedures guarantee successful back-translation from the clausal sets, derived with our query answering procedures, to a first-order formula. In general the back-translation problem is undecidable and often fails, nonetheless by our rules, this problem is solvable for Boolean conjunctive queries for all the considered guarded fragments. For practicality we use a saturation-based approach as the basis, so that all the procedures are well-primed for implementation in state-of-the-art modern first-order theorem provers in the future.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

This thesis would be impossible without the support of the following wonderful people.

I am deeply indebted to my supervisor Renate A. Schmidt for her knowledge, inspiration and enthusiasm throughout my PhD, particularly during the pandemic. Without her unequivocal support and guidance, this thesis would never have been completed. I express my warmest gratitude to my co-supervisor Giles Reger, for his valuable advice and support.

I am very appreciative of my examiners Konstantin Korovin and Pascal Fontaine for reviewing this thesis and providing constructive feedback. Special thanks to Uwe Waldmann for the comments on selection functions and Hans de Nivelle for the insights on deciding the guarded fragment. I would like to thank my MSc project supervisor Ning Zhang for her support and encouragement.

Many thanks to my past paper reviewers for their helpful comments. I wish to thank the Great Britain-China Educational Trust for the generous funding.

It is the following lovely people that make Manchester an engaging place for me to live. Thank you Chen Qian for tolerating my complaints in pubs every-time, thank you Haoruo, Heng and Rui for saving me when I am buried in work, thank you Ruba for your constant support and thank you Yizheng for your invaluable suggestions in surviving in academic. Thank you Albudula, Chai, Chen Li, David, Hiris, Hizal, Julio, Ghadah, Kiana, Meizhi, Mostafa, Warren, Xue Hua, Yangmei and the list is largely incomplete. I would like to thank all the friends I made in UK, in China and worldwide, who made the past years joyful and unforgettable. For people I forgot to mention, my apologies.

Last but not least, I thank my family for the moral and financial support, particularly my mother, for your unwavering love and belief in me.

I better stop here before these acknowledgements receive an immediate rejection for being overly long.

It is a pleasure to have this journey with you all. Thank you, for everything.

Chapter 1

Introduction

Developing *automated querying procedures* is an indispensable, yet challenging task in modern information systems. On the abstract level in these systems, rules and queries are commonly formalised as *first-order formulas*, hence the querying problem is indeed a reasoning task in *first-order logic*. In particular a querying procedure should be *sound*, *complete* and *practical*. Due to *undecidability* of first-order logic [Chu36, Tur36], ideally one wants to use *decidable* and *computationally well-behaved* fragments of first-order logic as a basis for querying tasks. In this thesis, we devise the first practical decision procedures for querying in arguably the most pioneering and robustly decidable fragments of first-order logic, namely a family of the *guarded first-order fragments*; see [ANvB98, Grä99a, Mar07, tCS13, BtCS15].

Why the guarded first-order fragments? In mathematical logic, a fundamental problem is checking whether an arbitrary first-order formula is *satisfiable*. In 1928 this problem was formalised as the *Entscheidungsproblem* [HA28, Page 77], literally meaning 'decision problem'. The first groundbreaking result is *Gödel's Incomplete Theorem* [Göd30, Göd31], techniques of which inspired Church and Turing to independently prove that first-order logic is *undecidable* [Chu36, Tur36]. Despite the negative effect of the *Church-Turing thesis*, the decision problem retains its vitality, being revised as a *classification problem* posing the question: in first-order logic, which fragments are decidable and which are not?

As early as 1915, first-order logic with *unary predicate symbols* was proved decidable, but not with *binary predicate symbols* [Löw15]. This result was then strengthened to *three* binary predicate symbols [Her31] and subsequently *one* binary predicate symbol [Kal37]. At the same period the *prefix quantifier classes*, of which quantifiers only occur at the outermost of formulas, are progressively discovered. Among others the *Bernays-Schönfinkel class* [BS28], the *Ackermann class* [Ack28] and the *Gödel class* [Göd32, Kal33, Sch34] were found to be decidable as well as many undecidable classes were identified [Gur65, Gur68, Gol84, GL75, Sur59]. By the early 1980s the classification task was by and large complete; see [DG79, BGG97, Lew79] for a comprehensive treatment and [Grä03] for a succinct discussion.

However from a practical perspective, the prefix quantifier classes are not suitable for computational purposes for lack of good model-theoretic properties. In contrast to the prefix quantifier classes, *propositional modal logic* [Pop94, BRV01a], because of its well-behaved *model-theoretic properties* [BvB07] and *robust decidability* [Var96], has been applied to various areas of computer science such as program verification [Pra80, CES86, Seg82], databases [dCCF82, Mar88, Fit00], artificial intelligence [BLMS94, MH69] and multi-agents systems [Lia03, HF89]; see also [BvBW07, Part 4]. Therefore, the possibility of first-order generalisation of modal logic was spotlighted.

The *two-variable fragment* of first-order logic [vB91, Gab81] can be seen as a first-order generalisation of modal logic, which has been proved to be decidable by reducing the decision problem of it to that of the *Gödel class*; see [Sco62]. Though the *Gödel class with equality* is undecidable [Gol84], the *two-variable fragment with equality* is decidable [Mor75]. Nevertheless, the decidability result of two-variable fragment is *not as robust as* that of modal logic, since with common properties such as transitivity the two-variable fragment becomes highly undecidable; see [GO99, GKV97, GOR99].

Eventually attention shifted to the *guarded fragment* [vB97, ANvB98]. Unlike all of the aforementioned decidable fragments, the guarded fragment is *decidable* [ANvB98], it has the *tree model property* and the *finite model property* like modal logic, and is thus *robustly decidable* [Grä99b]. Due to these hoped-for properties, the guarded fragment has been considered with numerous extensions: the guarded fragment with either *functionality*, or with *counting quantifiers* or with *transitivity axioms* is undecidable [Grä99b] but the *guarded fragment with transitive guards*, viz. the transitive predicate symbols appear only in the guard positions, is decidable [ST04]. The guarded fragment has also been merged with characterisations the two-variable fragment: the combination of the *guarded fragment* and the *two-variable fragment* is decidable [Kaz06], the *tri-guarded fragment* extending both the guarded and the two-variable fragments is decidable [RS18], the *two-variable guarded fragment with transitive relations* is undecidable [GMV99] and the *guarded two-variable fragment with counting quantifiers* is decidable [Pra07].

By relaxing the condition of the 'guard' literals, the guarded fragment extends to the *loosely guarded fragment* (the *pairwise guarded fragment*) [vB97] and the *clique guarded fragment* [Grä99a], which is also called the *packed fragment* [Mar07]. These fragments are called the *guarded quantification fragments* since the distinguishing 'guardedness' pattern is in *quantified formulas* of these fragments. If the 'guardedness' restriction is applied to the *negated formulas*, then one obtains the *guarded negation fragments*, consisting of the *unary negation fragment* [tCS13] and its extensions: the *guarded negation* and the *clique guarded negation fragments* [BtCS15]. All these guarded fragments are *robustly decidable* and they enjoy *well-behaved computational properties*; see also [Grä99b, BBtC18, HM02, BtCO12, BBtC13]. Further details on these guarded fragments are presented in **Section 2.1**.

In real-world applications, multiple restricted forms of the aforementioned guarded fragments have been used as logical formalism in several areas of computer science. In knowledge representation and semantics web *expressive description logic* $\mathcal{ALCHOI}$ and its fragments [BHLS17, BN07, CG07], which can be viewed as *guarded fragments with unary and binary predicate symbols*, are successfully applied to diverse areas such as medical informatics and natural language processing; see [HPMW07, Part III] and [HPvH03]. In the past twelve years, *Datalog*$^{\pm}$, an extension of *Datalog* [CGT89], has been developed as an expressive ontological language for querying purposes; see [CGL09, CGL$^+$10, CGL12]. In *Datalog*$^{\pm}$ rules, the *linear*, the *guarded* and the *frontier guarded Datalog*$^{\pm}$ *rules* are pinpointed for having nice computational properties as these rules are *Horn fragments of the guarded negation fragment*; see [BLMS11, BtCO12, GRS14]. *Datalog*$^{\pm}$ is also investigated in connection with *existential rules* and *tuple-generating dependency*.

These facts motivated us to develop practical decision procedures for satisfiability checking and querying for these guarded fragments.

Why the saturation-based procedures? Given an arbitrary problem, can one solve it by formalising the problem and then applying mechanical computations to the formalised axioms? This vision can be traced back to the *calculus ratiocinator* by Leibniz. After about three centuries this vision is eventually realised in the availability of automated theorem provers. Using mathematical logic as foundation, automated theorem provers productively build proofs for a given problem. Unlike *model-theoretic procedures*, automated provers are rooted in the *proof-theoretic tradition*, empowering machines to automatically or interactively solve problems, given as sets of formulas.

An important landmark in the development of automated provers is Robinson [Rob65a], inventing the combined use of unification and the resolution principle. In the same year many efficient and elegant techniques such as the *hyperresolution rule* [Rob65b] and the *set-of-support strategy* [WRC65] were created. Until now for many *practical reasoning tasks* the area has flourished with diverse advanced methods such as the *tableaux methods* [Häh01], the *inverse method* [DV01b], the *resolution calculus* [BG01], the *paramodulation calculus* [GR69] the *superposition calculus* [BG98] and the *sequent calculus* [DV01a] being developed. Among these techniques resolution and superposition are the core to saturation-based inference systems, on which state-of-the-art first-order automated theorem provers such as E [Sch13], Vampire [RV01b] and Spass [WDF$^+$09] are built. The foundation to these saturation-based provers are the powerful *resolution* and *superposition-based frameworks* of [BG01, BG98]. Currently automated theorem proving have been broadly applied to real-world applications such as problem solving [FN71, Gre69], software engineering [Sch01], verification [Har08, CRSS94, Moo10] and assisting mathematical proofs [NS56]; see also [Sut] and also [NML$^+$19] for a survey of theorem provers.

In the seminal work of [Joy76] resolution is used as a basis for *decision procedures* for several prefix quantifier classes. About 1990 the development of resolution-based (and superposition-based) decision procedures outbursts fruitful results for decidable classes of first-order clauses; see [HS99, FLTZ93, FLHT01] for comprehensive treatments and also [GHS02, SH00, dN00, BGW93]. Due to the many successful applications of *modal logic* and its close cousin *description logic* to computer science, practical resolution-based decision procedures have been developed for these logics; see [Hus99, AdRdN01, AdNdR99, FLHT01] for both modal logics and description logics; see also [Mot06, KM08, HMS08, Kaz06] for description logics and [HdNS00, Sch96, GHMS98, Sch98, Sch99, SH13, ZHD09, NDH19] for modal logics.

After 2000 attention gradually turned to developing practical decision procedures for the first-order generalisations of modal logic. Resolution-based procedures have been devised for the *two-variable fragment with equality* [dNP01] and a *restricted form of the guarded fragment*, viz. *GF1*$^-$ and its extensions [GHS03]. As for the *guarded fragments* we are interested in this thesis, resolution-based decision procedures have been devised for the *guarded fragment* [dNdR03, Kaz06], the *loosely guarded fragment* [dNdR03, ZS20a], the *guarded fragment with equality* [GdN99, Kaz06] and the *loosely guarded fragment with equality* [GdN99]; see also [KdN04] and [Kaz06] for investigation on deciding the *guarded fragment with transitive guards* and *transitive and compositional guards*, respectively. The tableau-based decision procedures were also developed for *GF1*$^-$ [LST99], the *guarded fragment* [Hla02] and the *clique guarded fragment with equality* [HT01]. At that time the development of practical decision procedures kept up with the hunt of new decidable first-order fragment. At present there exist however no practical decision procedures for the newly discovered *unary negation*, the *guarded negation* and the *clique guarded negation fragments*, not to mention the absence of practical decision procedures for querying in the *guarded*, the *loosely guarded*, the *clique guarded*, the *unary negation*, the *guarded negation* and the *clique guarded negation fragments*. This thesis aims to close this gap. Our methods are based on the resolution and superposition situated in the saturation-based frameworks of [BG01, BG98].

Why the targeted querying problems? *Conjunctive queries* [AHV95, Ull89], corresponding to *select-project-join queries* in relation algebras, enjoy prominent presence in the areas of database and knowledge presentation. *Boolean conjunctive queries* (*BCQ*s), also known as *positive existential queries*, are conjunctive queries without *answer variables* (*free variables*). The problem of answering conjunctive queries is generally understood as that of answering BCQs, since by instantiating the answer variables in conjunctive queries with constants in the database, the problem of conjunctive query answering can be reduced to that of BCQ answering in polynomial time. More importantly, vital problems such as query evaluation [CM77], query containment [CM77], constraint-satisfaction problems [KV00] and homomorphism problems [Var00], can be recast as BCQ answering problems.

Ontology-mediated querying, also called *ontology-based data access* (*OBDA*), is widely regarded as a key component of next generation information systems; see [PLC+08, CDGL+07, DFK+08, HMA+08] for its origins. Given (possibly incomplete) data D of multiple (possibly heterogeneous) databases and a query q, an OBDA system defines a global conceptual schema (i.e. a knowledge base or an ontology) Σ from the databases, so that with a new query Σ_q compiled from Σ and q, the problem of checking q over multi-schemas and cross-datatypes databases is reduced to a model checking problem $\mathrm{D} \models \Sigma_q$, which can be solved by highly-efficient SQL, Datalog or other database engines. OBDA systems are discussed in [CGL+11, PCS14, CCK+17, SM13, MGS+19]. Important works on query rewriting techniques with the ontologies generally expressible in the *considered guarded fragments* are [GOP14, AOS18, BBGP21] for *guarded Datalog*$^\pm$ and [CTS11, PHM09, CGL+07] for *description logics*. See [XCK+18, Kog12] for surveys on OBDA techniques and systems; [KRZ13] gives a tutorial on OBDA.

Unfortunately with arbitrary formulas Σ in any of *the guarded fragment and its extensions*, a union q of BCQs and datasets D, there may not exist a first-order formula (or a Datalog rule) Σ_q such that the entailment checking problem $\Sigma \cup \mathrm{D} \models q$ can be reduced to a model checking problem $\mathrm{D} \models \Sigma_q$. [BBGP21] gives a counter-example (Example 2.2) for the case of guarded Datalog$^\pm$. In this case Σ and q are said to be not *first-order* (*Datalog*) *rewritable* [CDGL+07]. For recent techniques and results on the first-order and the Datalog rewritings, the papers [BKK+18, HLPW18, TW21, KNG16, FKL19, AOS20] may be consulted.

Due to the negative result of *first-order rewritability* for *the guarded fragment and its extensions*, we propose novel settings of *saturation-based BCQ answering* and *rewriting* for the considered guarded fragments. The following two scenarios show the benefit of using saturation-based approaches for solving BCQ answering problems: deciding the entailment $\Sigma \cup \mathrm{D} \models q$ or equivalently checking unsatisfiability of $\{\neg q\} \cup \Sigma \cup \mathrm{D}$ with Σ formulas in any of the considered guarded fragments, q a union of BCQs and D databases.

1. Suppose Σ is fixed and N_1 is computed as the saturation of Σ. With constantly updated q and D, N_1 can be reused in saturating $\{\neg q\} \cup N_1 \cup \mathrm{D}$ to avoid repeated inference steps in saturating N_1, thus accelerating the querying processes.

2. Suppose both Σ and q are fixed. Different to the case of Scenario 1., here

it makes sense to pre-saturate $\{\neg q\} \cup \Sigma$. If N_2 is this pre-saturation, then regardless as to whether adding, deleting or updating datasets D, N_2 can be reused to prevent recomputing numerously inferences unnecessarily in checking the satisfiability of $N_2 \cup \mathrm{D}$.

Next we motivate the our saturation-based BCQ rewriting problem. Suppose N_3 is a clausal set produced by saturating $\{\neg q\} \cup \Sigma$. We propose an attempt to back-translate (and then negate) N_3 into a *first-order formula* F such that $\Sigma \cup \mathrm{D} \models q$ if and only if $\mathrm{D} \models F$. F is then a first-order formula or even a (*clique*) *guarded formula* that gives user an explicit view of the querying process. The saturation-based rewritings have potential to be useful for *query explanation*. Most importantly devising the back-translation procedures is interesting and challenging in its own right, as in general it is an undecidable problem and often fails.

The problem of BCQ answering for ontologies is traditionally handled by database techniques such as the *chase algorithm* [ABU79, MMS79] (also known as *materialisation*), and the *forward and backward chaining* techniques [RN20, Chapter 7]. Versatile as automated theorem provers are, they have insufficiently used as query engines. Hence, we are interested to see how automated reasoning techniques handle BCQ answering and rewriting problems, especially how saturation-based decision procedures can be developed to solve conventional querying problems.

Challenges

The focuses of the thesis are the following two problems.

i. **BCQ answering**: Given a set Σ of first-order formulas in any of the considered guarded fragments, a dataset D and a union q of BCQs, we determine whether $\Sigma \cup \mathrm{D} \models q$, viz. test if $\{\neg q\} \cup \Sigma \cup \mathrm{D}$ is unsatisfiable.

ii. **Saturation-based BCQ rewriting**: As the saturation-based frameworks are based on first-order clauses, saturations $\{\neg q\} \cup \Sigma \cup \mathrm{D}$ represented as clausal sets. The second problem we are interested in is the saturation of the set $\{\neg q\} \cup \Sigma$ and its back-translation into a first-order formula Σ_q such that $\Sigma \cup \mathrm{D} \models q$ if and only if $\mathrm{D} \models \Sigma_q$.

Problems i. and ii. are formally defined in **Section 2.2**.

To use saturation-based methods to address i. and ii. the first and foremost task is devising saturation-based inference systems that are sound and refutationally complete. The next main task is to develop refinements to ensure termination on all input problems. As our procedures are in line with either resolution [BG01] or superposition-based framework of [BG98], for termination the following properties must hold.

1. The *depth*, viz. the nesting number of compound terms, of any derived clauses is finitely bounded.
2. The *width*, viz. the number of distinct variables, in any derived clause, is finitely bounded.
3. In any derivation the number of symbols in the signature is finitely bounded.

In a saturation-based derivation, Properties 1.–3. can be ensured if the conclusions are no deeper and no wider than at least one of its premises and only finitely many signature symbols are needed. Property 2. above assumes that clauses are *condensed* [NW01] and are identical modulo *variable renaming*.

For conciseness we use the notations BCQ and FOL to represent the *Boolean conjunctive query* and *first-order logic*, respectively, and use $\mathsf{BCQ}_{\approx}$ and $\mathsf{FOL}_{\approx}$ to denote *BCQ with equality* and *FOL with equality*, respectively. Further we use the notations GF, LGF, CGF, UNF, GNF and CGNF to denote the *guarded*, the *loosely guarded*, the *clique guarded*, the *unary negation*, the *guarded negation* and the *clique guarded negation fragments*, respectively. In general when we say BCQ, we mean BCQ when the querying fragments are one of the guarded quantification fragments (as equality is not allowed), otherwise we mean $\mathsf{BCQ}_{\approx}$. Note that UNF is a special case of GNF with only equality literals as the 'guard' literals, therefore all results established for GNF immediately hold for UNF. Hence this thesis does not particularly discuss the querying procedures for UNF. All the aforementioned fragments are surveyed in **Section 2.1**.

To solve the two main problems i. and ii. of interest the following challenges need to be tackled.

1) Devising saturation-based decision procedures for checking satisfiability of GF, LGF, CGF, GNF and/or CGNF.
2) Handling BCQs with the presence of GF, LGF, CGF, GNF and/or CGNF.
3) Finely combining the procedures for 1) and 2) to solve BCQ answering for GF, LGF, CGF, GNF and/or CGNF.

4) Back-translating the clausal sets obtained by the procedures developed in 3) to a first-order formula, thereby obtaining saturation-based BCQ rewriting procedures for GF, LGF, CGF, GNF and/or CGNF.

We separately discuss how Challenges 1)–4) have been tackled in this thesis.

Deciding the guarded first-order fragments The first challenge is developing saturation-based decision procedures for the considered guarded fragments. For each fragment it requires us to address three tasks, namely devising a clausification process, developing a saturation-based inference system and proving a termination result.

We first develop the clausification processes for BCQs and the considered guarded fragments. For GF and LGF we devise the clausification process $\mathbf{Trans}^{\mathsf{GF}}$, transforming formulas into *guarded* and *loosely guarded clauses* (*LG clauses*), respectively. By rigorously investigating the guardedness patterns in CGF, GNF and CGNF, we devise three innovative clausification processes, namely the $\mathbf{Trans}^{\mathsf{CGF}}$, the $\mathbf{Trans}^{\mathsf{GNF}}$ and the $\mathbf{Trans}^{\mathsf{CGNF}}$ processes, so that CGF, GNF and CGNF are transformed to *LG clauses*, *guarded clauses with equality and query clauses with equality* (*$GQ_{\approx}$ clauses*) and *loosely guarded clauses with equality and query clauses with equality* (*$LGQ_{\approx}$ clauses*), respectively. Our clausification transforms a union of BCQs to *query clauses* and *query clauses with equality* (*$Q_{\approx}$ clauses*). The class of $\mathsf{LGQ}_{\approx}$ clauses can be seen as the combination of *loosely guarded clauses with equality* (*$LG_{\approx}$ clauses*) and *$Q_{\approx}$ clauses*. Figure 1.1 summaries the way that the formulas from different guarded fragments and BCQs are transformed into their respective types of clauses.

Next, we devise the saturation-based inference systems in accordance with either the resolution-based framework of [BG97, BG01] or the superposition-based framework of [BG98]. Our inference systems aim to decide satisfiability of the classes of *guarded*, *LG*, *$GQ_{\approx}$* and *$LGQ_{\approx}$ clauses* associated with the targeted fragments. Unlike conventional saturation-based systems (such as the **Satu** and $\mathbf{Satu}_{\approx}$ systems presented in **Section 3.4**), our inference systems make use of two innovative techniques: the *partial selection-based resolution rule* (the ***P-Res** rule*) and the *top-variable resolution refinement*.

1. The **P-Res** rule is critical to our systems. Whenever the *standard selection-based ordered resolution rule* (the ***Res** rule*) is applicable to a clause C (as the main premise) and clauses $C_1, \ldots, C_n$ (as the side premises), one

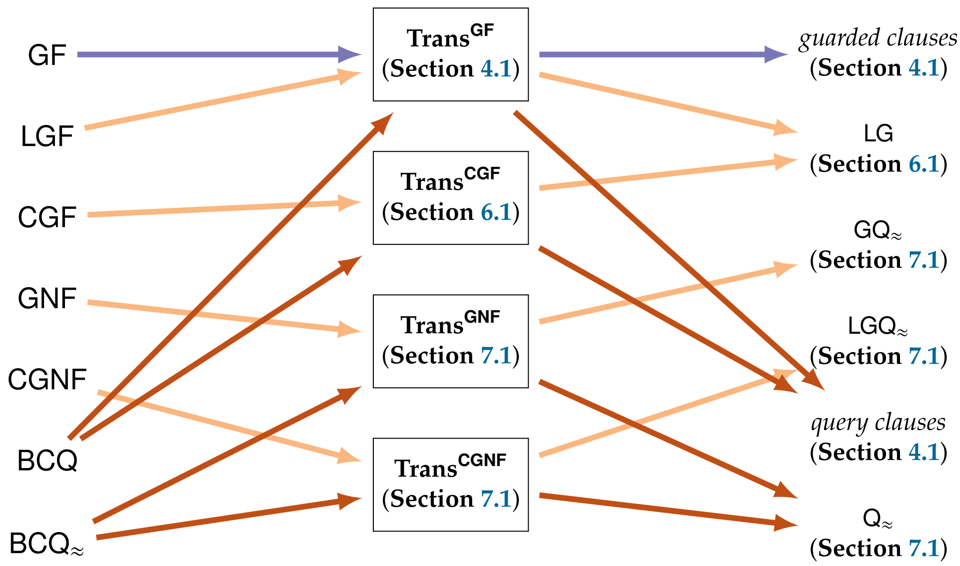

Figure 1.1: The relationship of the targeted fragments, the customised clausification processes and the obtained clausal classes

can apply the **P-Res** rule to C (as the main premise) and a *subset of* $C_1, \ldots, C_n$ (as the side premises) instead. With the same main premise as the **Res** rule, a **P-Res** inference step allows any subset of the **Res** side premises to be its side premises, thus gives us the flexibility to derive only the desirable **P-Res** resolvents from all the possible **P-Res** resolvents.

Given any sound and refutationally complete saturation-based resolution inference system, its resolution rule can be safely replaced by the **P-Res** rule. In this thesis we develop the ***P-Res*** *inference systems* **Inf** and $\mathbf{Inf}_{\approx}$, for first-order clausal logic without and with equality, respectively.

2. For the **P-Res** rule, we devise the *top-variable resolution refinement* so that in an **P-Res** inference step the chosen **Res** side premises contain the potentially deepest terms. For the considered clausal classes, this refinement effectively avoids term depth increase in the **P-Res** resolvents, ideally satisfying Property 1. crucial for having termination.

   Based on the top-variable refinement, we define the $\mathbf{T\text{-}Ref}^{\mathsf{GQ}}$, the $\mathbf{T\text{-}Ref}^{\mathsf{LGQ}}$ and the $\mathbf{T\text{-}Ref}_{\approx}^{\mathsf{LGQ}_{\approx}}$ refinements. These refinements and the **P-Res** inference systems provide the basis for the *top-variable resolution systems* $\mathbf{T\text{-}Inf}^{\mathsf{GQ}}$, $\mathbf{T\text{-}Inf}^{\mathsf{LGQ}}$ and the *top-variable superposition system* $\mathbf{T\text{-}Inf}_{\approx}^{\mathsf{LGQ}_{\approx}}$ we devise. All are proved sound and refutationally complete for first-order

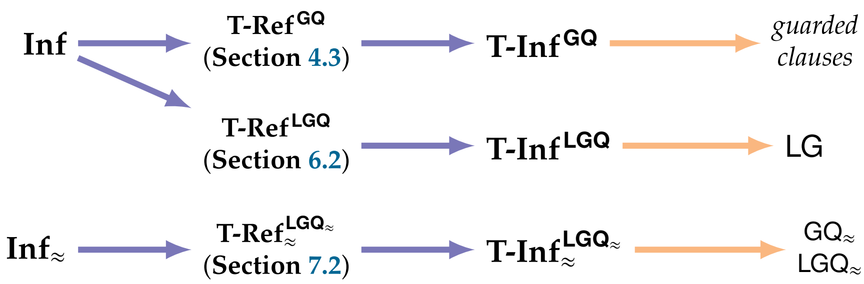

Figure 1.2: The relationship of the newly devised inference systems and the related clausal classes

clausal logic (with equality). Figure 1.2 describes the relationship of the **P-Res** systems, the top-variable-based refinements (and the sections when they are presented), the top-variable inference systems and the relevant clausal classes.

The **P-Res** rule and the top-variable technique are presented in **Sections** 4.2 and 4.3, respectively. The inference systems presented in this thesis are exhibited in Figure 1.3 with the sections where they can be found.

The last task is proving that for the aforementioned guarded clausal classes, the top-variable inference systems are *guaranteed to terminate*. With termination established, these systems provide decision procedures for our guarded clausal classes due to the fact that these systems are sound and refutationally complete for first-order clausal logic (with equality). We formally prove application of the **T-Inf**$^{\mathsf{GQ}}$, **T-Inf**$^{\mathsf{LGQ}}$ and the **T-Inf**$_\approx^{\mathsf{LGQ}_\approx}$ systems, respectively, to classes of guarded, LG and LG$_\approx$ clauses derive only guarded, LG and LG$_\approx$ clauses with bounded width. Hence the **T-Inf**$^{\mathsf{GQ}}$, **T-Inf**$^{\mathsf{LGQ}}$ and the **T-Inf**$_\approx^{\mathsf{LGQ}_\approx}$ systems decide satisfiability of the guarded, LG and LG$_\approx$ clauses classes, respectively.

	resolution-based (for FOL)	superposition-based (for FOL$_\approx$)
basic inference systems	**Satu** (**Section** 3.4)	**Satu**$_\approx$ (**Section** 3.4)
the **P-Res** inference systems	**Inf** (**Section** 4.2)	**Inf**$_\approx$ (**Section** 7.2)
the top-variable inference systems	**T-Inf**$^{\mathsf{GQ}}$ (**Section** 4.3) **T-Inf**$^{\mathsf{LGQ}}$ (**Section** 6.2)	**T-Inf**$_\approx^{\mathsf{LGQ}_\approx}$ (**Section** 7.2)

Figure 1.3: A classification of the provided inference systems

Roughly speaking in our top-variable refinements, we adopt the principle that the eligible literals in a clause are *the deepest and the widest literals*, one of which is the key to ensure termination. **Sections** 4.4, 6.3 and 7.3 present how satisfiability of the guarded, LG and $\mathsf{LG}_{\approx}$ clausal classes are can be decided.

Handling BCQs The second main challenge is the handling of the given union of BCQs. In the previously discussed clausification processes, a union of BCQs is transformed to *query clauses* and *$\mathsf{Q}_{\approx}$ clauses* (i.e., *query clauses with equality*). In the conclusions of query and $\mathsf{Q}_{\approx}$ clauses, one needs to ensure that no unbounded depth or width increase occurs. For this goal we introduce new techniques, concisely discussed as follows.

I. For *$\mathsf{Q}_{\approx}$ clauses with inequality literals occurring* we use the *equality resolution rule* (the ***E-Res*** *rule*). By the carefully devised *superposition refinement*, it is ensured that in our inference systems only the **E-Res** rule is applicable to $\mathsf{Q}_{\approx}$ clauses, so that applying rules in the *top-variable systems* to $\mathsf{Q}_{\approx}$ clauses solely derives $\mathsf{Q}_{\approx}$ clauses and *query clauses*.

II. In *query clauses* occurrences of variables are unrestricted, thus analysing the conclusions of these clauses is difficult. To dissect variables in query clauses, we create two novel *separation rules* and a goal-oriented *query separation procedure* (the ***Q-Sep*** *procedure*). The **Q-Sep** procedure is crucial to control the computations of inferences on query clauses. By this procedure, a query clause Q is replaced by an equisatisfiable set N of less wide *inseparable query clauses* and *Horn guarded clauses* (*HG clauses*).

   By our definitions the inseparable query clauses are formally defined as *indecomposable chained-only query clauses* (*indecomposable CO clauses*), which enjoy a key property: in these query clause each variable 'chains' at least two distinct literals.

III. We use the *top-variable resolution rule* to compute the conclusions of *indecomposable CO clauses* with in the presence of *guarded*, *LG* and *$\mathsf{LG}_{\approx}$ clauses*. In this resolution computation, an indecomposable CO clause is the *main premise* and guarded clauses (respectively LG and $\mathsf{LG}_{\approx}$ clauses) are the *side premises*. By the top-variable technique, the derived *top-variable resolvent* is guaranteed to be no deeper than at least one of its premises. However, the resolvent can be wider than all of its premises.

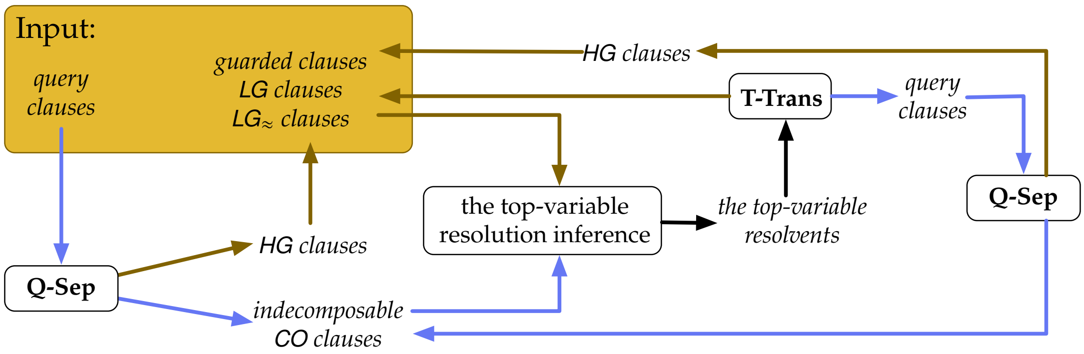

Figure 1.4: Handling query clauses in the presence of studied clausal classes

IV. For the *top-variable resolvents* R of *indecomposable CO clauses* in the presence of *guarded*, *LG* and *LG*$_\approx$ *clauses*, we devise a sophisticated form of *structural transformation* (the ***T-Trans*** *rule*) so that R is transformed into an equisatisfiable set N of *query clauses* and *guarded*, *LG* and *LG*$_\approx$ *clauses*, respectively. In particular in N each clause is no wider than at least one of its top-variable premises. The derived query clauses are coped with by the **Q-Sep** procedure, and the derived guarded, LG and LG$_\approx$ clauses are handled by their respective top-variable inference systems.

The results of I.–II. are presented in **Sections** 7.3 and 4.5, respectively. For guarded, LG and LG$_\approx$ clauses, the rest of results are discussed in **Sections** 4.5, 6.3 and 7.3, respectively. Figure 1.4 is a flow chart of the query handling procedure presented in II.–IV.

Devising BCQ answering procedures With the *query handling procedures* and the *decision procedures for the targeted guarded fragments* created, the next main challenge is to properly amalgamate these procedures, give us the sought *decision procedures for answering BCQs in the targeted guarded fragments*.

Integrating the query handling processes into saturation-based systems poses new challenges. Once two procedures are combined, new predicate symbols (introduced in handling queries) occur in the saturation, hence for the termination results we need to ensure *only finitely many* of these symbols are introduced. We formally prove that if we *reuse the existing introduced predicate symbols* to define clauses that are identical modulo variable renaming, in the saturation only finitely many new predicate symbols are required. This result is based on the facts that newly introduced clauses have a bounded number

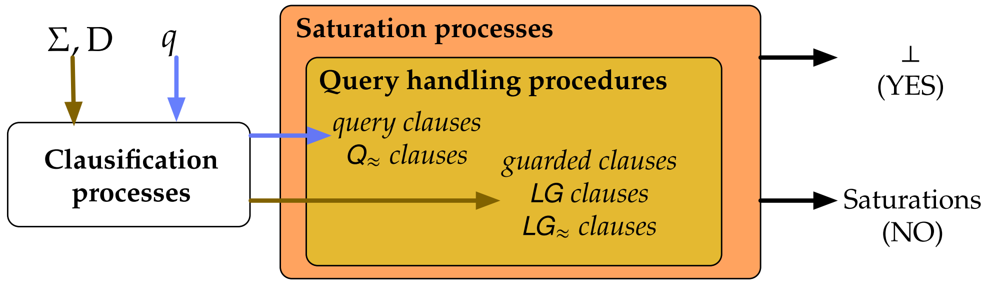

Figure 1.5: The saturation-based query answering procedure

of variables, and these clauses are composed of the signature symbols before saturation processes.

The next task considers the *inference steps* in the saturation for *query clauses* and *Q≈ clauses*. As said above, for Q≈ clauses we devise *superposition refinement* so that only the **E-Res** rule is applicable to these clauses. In query clauses only the *indecomposable CO clauses* derives conclusions, therefore for these clauses we devise an appropriate *resolution refinement*, so that it is guaranteed that only the *top-variable resolution rule* is applicable to the indecomposable CO clause with itself being the main premise and guarded clauses (LG and LG≈ clauses thereof) being the side premises.

The final task requires us to formally present the query answering procedures in the saturation-based framework. Though our procedures do not rely on a particular form of saturation processes, we devise the query answering procedures in accordance with the *given-clause algorithm* in [Wei01, MW97], since this algorithm has been implemented as a basis for modern first-order theorem provers such as Spass [WDF+09], Vampire [RV01b] and E [Sch13]. This choice ensures the implementation of our procedures is practical and approachable.

Suppose Σ are formulas in one of our guarded fragments, q is a union of BCQs and D is a set of ground atoms. To check whether $\Sigma \cup \mathrm{D} \models q$, we transform $\{\neg q\} \cup \Sigma \cup \mathrm{D}$ to a clausal set N. If N is unsatisfiable, then it is the case that $\Sigma \cup \mathrm{D} \models q$, otherwise $\Sigma \cup \mathrm{D} \not\models q$. Figure 1.5 illustrates our decision procedures for answering q in Σ and D. The *decision procedures for answering BCQs for GF* (the ***Q-Ans***$^{\mathsf{GF}}$ *procedure*), *LGF/CGF* (the ***Q-Ans***$^{\mathsf{CGF}}$ *procedure*) and *GNF/CGNF* (the ***Q-Ans***$^{\mathsf{CGNF}}$ *procedure*) are presented in **Sections 4.6, 6.4** and **7.4**, respectively.

Devising saturation-based BCQ rewriting procedures Finally we address Challenge 4): back-translating the clausal set, produced by the previous BCQ answering procedures, to a *first-order formula*. The target of Challenge 4) is stronger than that of main problem ii. since this challenge aims to back-translate a *derivation*, not necessarily a *saturation*.

For the considered guarded clausal classes the back-translation task is not straightforward. For example it is impossible to back-translate the *guarded clause* $\neg G(x, y) \vee A_1(f(x, y)) \vee A_1(f(y, x))$ to a first-order formula, due to the co-occurrences of the compound terms $f(x, y)$ and $f(y, x)$. In fact a clausal set N can be successfully back-translated to a first-order formula if N is *normal*, *unique*, *globally consistent* and *globally linear* [Eng96]. Based on the these prerequisites we devise our back-translation procedures. To avoid ambiguity the word *compatible* is used to replace the word *consistent*.

By investigating the applications of our clausification processes to GF, LGF, CGF, GNF and CGNF, we realise from these fragments the clausal classes have a nice property, viz. the *strong compatibility property*, which requires that the argument lists of all compound terms on one clause are identical. These clauses are the *aligned clauses*. To be specific the problem of answering BCQs for GF, for LGF/CGF and for GNF/CGNF is reduced to deciding satisfiability of the class of *query clauses and aligned guarded clauses* (*GQ*$^-$ *clauses*, consisting of *query clauses* and *G*$^-$ *clauses*), of *query clauses and aligned loosely guarded clauses* (*LGQ*$^-$ *clauses*, consisting of *query clauses* and *LG*$^-$ *clauses*) and of *query clauses with equality and aligned loosely guarded clauses with equality* (*LGQ*$^-_\approx$ *clauses*, consisting of *Q*$_\approx$ *clauses* and *LG*$^-_\approx$ *clauses*), respectively.

Next, we formally prove that our query answering procedures as well decide satisfiability of the classes of GQ$^-$, LGQ$^-$ and LGQ$^-_\approx$ clauses. Notably these

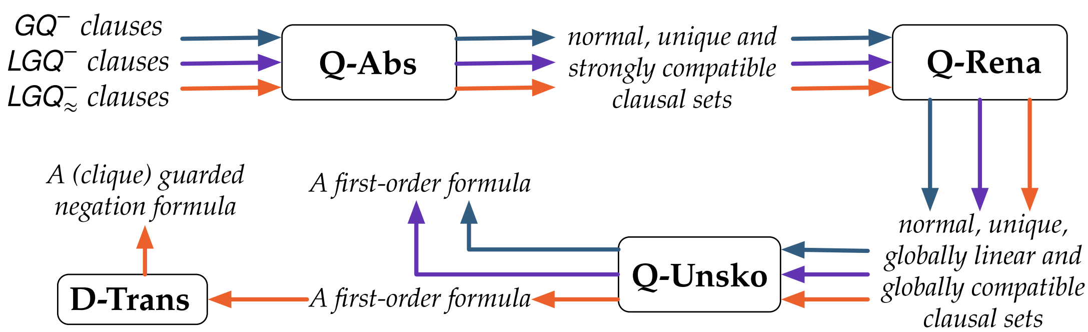

Figure 1.6: The back-translation procedure

clausal classes are each closed under the application of these query answering procedures. To back-translate GQ^-, LGQ^- and/or $\mathsf{LGQ}^-_{\approx}$ clausal sets to a first-order formula, one first needs to transform these clausal sets into *logically equivalent* normal, unique, globally compatible and globally linear clausal sets. Based on the *term abstraction* and *variable renaming rules* in [Eng96, GSS08a], we devise customised rules (the *constant and variable abstraction procedure* (the ***Q-Abs*** *procedure*) and the *variable renaming procedure* (the ***Q-Rena*** *procedure*)), so that GQ^-, LGQ^- and/or $\mathsf{LGQ}^-_{\approx}$ clausal sets are ensured to be transformed into a clausal set N satisfying the mentioned pre-requisites for successful back-translation. By our customised *unskolemisation procedure* (the ***Q-Unsko*** *procedure*), N is ensured to be unskolemised into a first-order formula. Unlike the classes of GQ^- and LGQ^- clauses, the $\mathsf{LGQ}^-_{\approx}$ clausal class is defined with the *protect property*, so that by a *special transformation* (the ***D-Trans*** *procedure*), the first-order formula F (back-translated from an $\mathsf{LGQ}^-_{\approx}$ clausal set) is reformulated as a *clique guarded negation formula*. The back-translation of GQ^- and LGQ^- clauses is not ensured to be reformed as formulas in GF, LGF and/or CGF, due to the fact that our back-translation procedures may introduce equality, which is not allowed in GF, LGF and/or CGF. Figure 1.6 describes the back-translation procedures for the targeted aligned clausal classes.

The *decision procedures for the saturation-based BCQ rewriting for GF* (the ***Q-Rew***$^{\mathsf{GF}}$ *procedure*), *LGF/CGF* (the ***Q-Rew***$^{\mathsf{CGF}}$ *procedure*) and *GNF/CGNF* (the

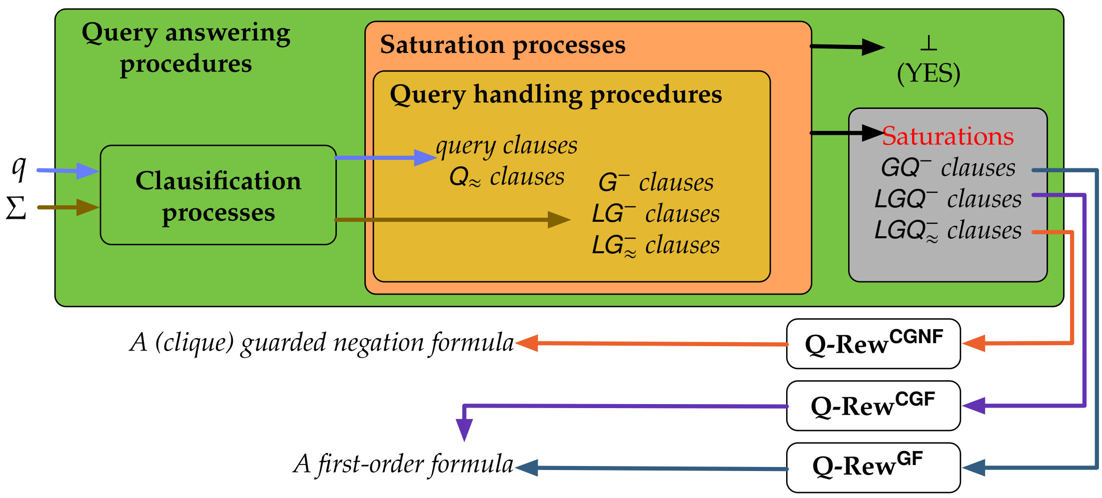

Figure 1.7: The saturation-based query rewriting procedure

Q-Rew$^{\textsf{CGNF}}$ *procedure*) are presented in **Chapter 5**, **Sections 6.4** and **7.4**, respectively. Figure 1.7 gives a complete view of the saturation-based query rewriting procedures with q a union of BCQs and Σ formulas in the fragments considered in this thesis.

Figure 1.8 on the next page summarises the relationships between the studied guarded fragments, queries, and clausal classes, in which an upper node is strictly more expressive than the adjacent one below it.

Contributions

The main contributions of thesis are:

1. We give the *first resolution-based decision procedure for CGF*, and the *first practical decision procedures for UNF, GNF and/or CGNF*.

2. We devise the *first practical decision procedures for answering BCQs for GF, LGF, CGF, UNF, GNF and/or CGNF*. These procedures provide practical solutions to arguably the *most difficult decision problems currently open in first-order logic* (*with equality*).

3. We develop the *first practical decision procedures for saturation-based BCQ rewriting in GF, LGF, CGF, UNF, GNF and/or CGNF*. In general back-translating a clausal set to a first-order formula is an *undecidable problem* and *often fails*, however by our *clausification processes*, *saturation procedures* and *back-translation techniques*, for the clausal classes we define it is ensures that the clausal sets can be *back-translated to a first-order formula*. The clausal sets of GNF and CGNF are even ensured to be *back-translated to a (clique) guarded negation formula*. These are interesting results.

4. We devise innovative and elegant ***P-Res*** *inference systems* and *top-variable inference systems*. These systems are *robust* as they are formally proved to be *sound* and *refutationally complete for general first-order clausal logic* (*with equality*), not just our clausal classes. These systems are well-prepared to provide *good foundations of practical decision procedures* for checking satisfiability for function-free fragments of first-order logic (with equality). With suitable clausification processes, the top-variable technique *guarantees to avoid term depth increase* in the conclusions (although in general this

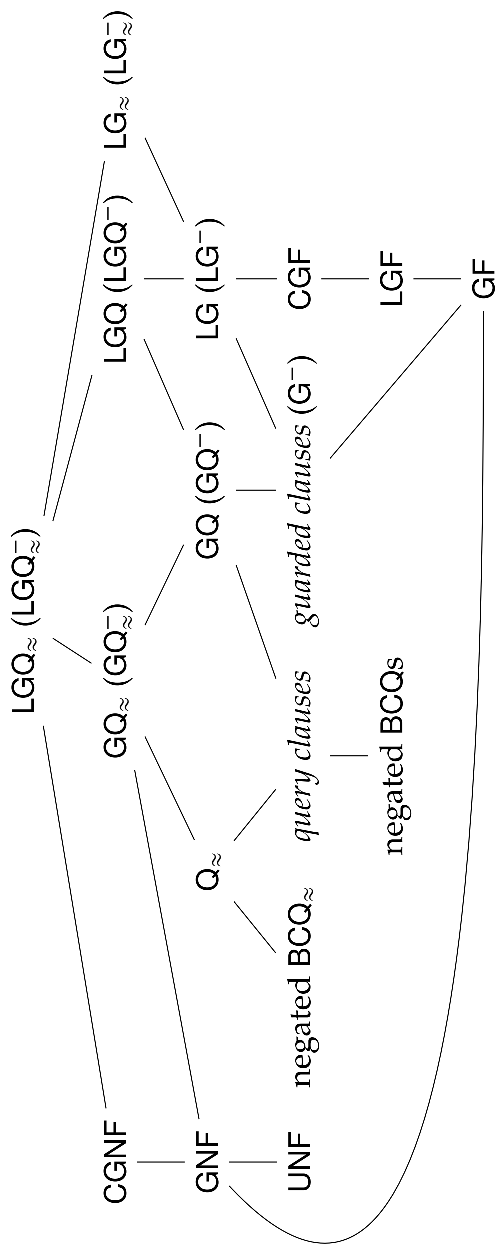

Figure 1.8: Relationships between the studied clausal classes and fragments

technique does not guarantee that the conclusions is no wider than its premises).

5. We devise *original automated reasoning techniques* that may advance the development of saturation-based theorem proving. The techniques include but are not limited to the *customised separation rules and goal-oriented query handling procedures*, the *novel clausification processes*, and the *back-translation rules and procedures*.

6. Our inference systems are *modular*. For any of our inference systems, by either *removing some refinement* or *adding a new refinement* (*satisfying the minimal requirements of admissible orderings and selection functions*), the system remains sound and refutationally complete for first-order clausal logic (with equality). In addition *simplification rules* and *redundant elimination techniques* (compatible with the saturation framework of [BG01]) are freely allowed in any of our systems.

More broadly the contributions of this thesis are:

1. We take a step forwards *bridging the gap between automated reasoning and databases*. In the area of databases saturation-based procedures are not typically applied as querying methods, as there are more well-developed techniques (such as the chase algorithm [ABU79, MMS79] and its variations) for database querying problems. Nevertheless due to the successful applications of highly-tuned automated theorem provers to real-world problems, it would be interesting to see how automated reasoners can be developed as query engines for solving real-world database querying problems. This direction of research has also been suggested in the database community [BKM$^+$17].

2. Our saturation-based query answering and rewriting procedures are well-equipped to provide the foundation for *other querying applications* such as query explanation. Our query rewriting procedures produce a *Skolem-symbol-free formula* representing a derivation, thus allows users to abstract explicit information during proof search of the given queries and formulas.

3. Our procedures are primed for *querying in real-world ontological languages* such as *guarded Datalog$^\pm$* and *frontier guarded Datalog$^\pm$* [CGK13, BLMS11]

and the *expressive description logic* $\mathcal{ALCHOI}$ [BHLS17] since these languages can be embedded in the considered guarded fragments.

Organisation

This thesis is organised as follows.

- **Chapter 2** formally defines all of the *considered guarded fragments*, the *saturation-based* BCQ *answering and rewriting problems* and summarises the known results for these fragments.

- **Chapter 3** gives both basic and customised notions of *first-order logic*, and the fundamentals of *saturation-based theorem proving*.

- **Chapter 4** first presents the ***P-Res*** and the *top-variable* resolution inference systems, and then devises a *saturation-based decision procedure for answering a union of* BCQ*s for* GF, which is the first main contribution of this thesis. A part of the result in this chapter is published in
  [ZS20b] Sen Zheng and Renate A. Schmidt. Querying the guarded fragment via resolution (extended abstract). In *Proc. PAAR'20*, volume 2752 of *CEUR Workshop Proceedings*, pages 167–177. CEUR-WS.org, 2020.

- **Chapter 5** devises the *decision procedure for saturation-based* BCQ *rewriting in* GF. Notably we define a refined clausal form of GF, namely the *aligned guarded clauses*, which by our customised rules, is guaranteed to be back-translatable to a first-order formula.

- In **Chapter 6**, we develop the *decision procedures for* BCQ *answering and saturation-based* BCQ *rewriting in* LGF *and/or* CGF. Unlike the procedures in **Chapters 4** and **5**, in this chapter our procedures particularly cope with the *loose* and the *clique guards* in LGF and CGF, respectively. The result of answering BCQs for the Horn fragment of LGF is published in
  [ZS20a] 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.

- **Chapter 7** is dedicated to devising the BCQ *answering and saturation-based* BCQ *rewriting procedures for* GNF *and/or* CGNF. Due to the occurrence of equality and inequality literals in GNF and CGNF, a novel superposition-based top-variable inference system is devised. Furthermore we identify a more sophisticated clausal form, viz. *aligned* (*loosely*) *guarded clauses with equality*, which comes with the assurance of being back-translatable to a (*clique*) *guarded negation formula*.

- **Chapter 8** discusses related work in three respects: existing resolution-based or superposition-based decision procedures for GF and LGF, current advancement for answering query in the fragments of the considered fragments and known query rewriting techniques.

- The last chapter concludes the thesis and suggests directions for future work.

Chapter 2

The guarded fragments and the querying problems

2.1 The guarded first-order fragments

The *guarded fragment* (*GF*) and the *loosely guarded fragment* (*LGF*) are introduced in [vB97, ANvB98], characterised as *modal fragments* of *first-order logic* (*FOL*). By the standard translation [BRV01b], modal formulas are translated into *guarded formulas*, where all quantified variables are 'guarded' by an atom. LGF, occasionally referred to as the *pairwise guarded fragment* [vB97, AMdNdR99], properly extends GF such that temporal operators [RU12] `until` and `since` can be expressed. Roughly speaking, in a *loosely guarded formula* all quantified variables are pairwise 'guarded' by a conjunction of atoms, namely a *loose guard*, in which the quantified variables form a 'clique'. Further LGF is extended to the *clique guarded fragment* (*CGF*) [Grä99a] such that existential quantifications are allowed in the loose guard, converting the 'clique' for the quantified variables to a branched 'clique' with branches made of the existential quantified variables in the loose guard. In [Hod02, Mar07] CGF is called the *packed fragment*. The findings of GF, LGF and CGF are based on the observation that all quantified formulas are relativised to (a conjunction of existentially quantified) atoms, hence the aforementioned guarded fragments are also called the *guarded quantification fragments*.

In the recent proposals, the guardedness pattern is associated with the negated formulas, leading to the *unary negation fragment* (*UNF*) [tCS13], the

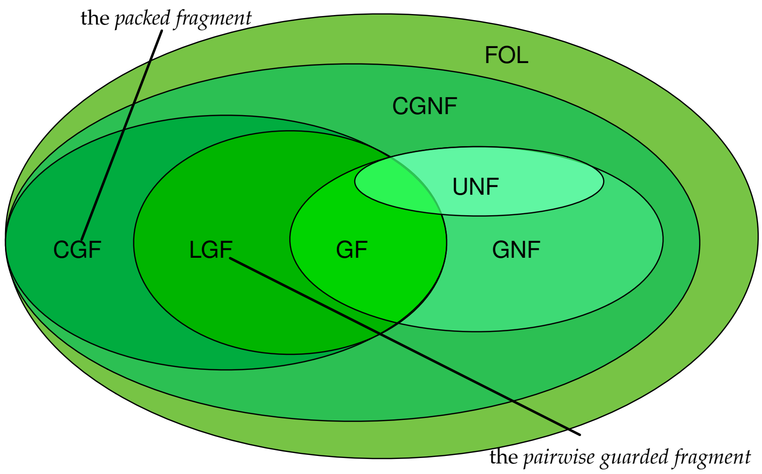

Figure 2.1: The relationship of the considered guarded fragments and FOL

guarded negation fragment (*GNF*) [BtCS15] and the *clique guarded negation fragment* (*CGNF*) [BtCS15], all of which are called the *guarded negation fragments* thereof. Unlike GF, UNF orthogonally generalises modal logic such that the negated formulas only have one free variable, thus UNF and GF are incomparable in terms of expressive power. GNF on the other hand, is more expressive than GF, since every *guarded sentence* can be represented as a *guarded negation sentence* [BtCS15]. In GNF all free variables of negated formula must be 'guarded' by an atom, and if that atom is an inequality literal $x \not\approx x$, GNF reduces to UNF. CGNF adopts the notion of 'clique' from CGF, extending GNF by allowing all free variables of the negated formulas to be pairwise 'guarded' by a conjunction of existentially quantified atoms. An informative introduction for the *guarded negation fragments* can be found in [Seg17]. Figure 2.1 presents the relationship between the aforementioned guarded fragments and FOL.

Both the *guarded quantification* and the *guarded negation fragments* are *robustly decidable* [Var96], meaning that these fragments have the *finite model property*, viz. every satisfiable formula has a finite model, and the *tree-like model property*, viz. if a formula has a model, then it has one of bounded tree width; see references in Figure 2.2. Satisfiability checking for any of *guarded quantification fragments* is 2ExpTime-complete, and is reduced to ExpTime-complete if a fragment has a *fixed signature* [Grä99b], however regardless of fixed signatures, checking satisfiability for any of *guarded negation fragments* is 2ExpTime-complete [tCS13, BtCS15].

	Guarded quantification fragments			Guarded negation fragments		
	GF	LGF	CGF	UNF	GNF	CGNF
Decidability	✓ [vB97] [ANvB98]	✓ [vB97]	✓ [Grä99b] [Mar07]	✓ [tCS13]	✓ [BtCS15]	
Satisfiability checking	EXP [Grä99b]	EXP [Grä99b]	EXP [Grä99b] [Mar07]	2EXP [tCS13]	2EXP [BtCS15]	
Tree-like model property	✓ [Grä99b]	✓ [Grä99a]	✓ [Grä99a]	✓ [tCS13]	✓ [BtCS15]	
Finite model property	✓ [Grä99b]	✓ [Hod02]	✓ [Mar07] [Hod02]	✓ [tCS13]	✓ [BtCS15]	
Craig interpolation	✗ [HM02]			✓ [tCS13]	✓ [BtCS15]	
Uniform interpolation	✗ [HM02]			open		

Figure 2.2: Interesting properties of the considered guarded fragments

We briefly discuss the *uniform interpolation* problem for these guarded fragments. A logical fragment S is said to have the *Craig interpolation property* [Cra57a, Cra57b] if F_1 and F_2 are two formulas in S such that $F_1 \models F_2$, then in S there exists a formula F expressed using only the common symbols of F_1 and F_2 such that $F_1 \models F$ and $F \models F_2$. A fragment S has the *uniform interpolation property* [Hen63] if for any S-formula F and a set of predicate symbols Δ, there is an S-formula F' with its symbols occurring in Δ such that $F \models F'$ and F is the strongest such entailment. Uniform interpolation entails Craig interpolation, but not vice-versa. The *guarded quantification fragments* do not have the Craig interpolation property, and hence also not the uniform interpolation property [HM02]. The *guarded negation fragments* have the Craig interpolation property [tCS13, BtCS15], yet it is unknown whether any the *guarded negation fragments* have the uniform interpolation property. For a restricted form of uniform interpolation (the *uniform modal interpolation*) for GF, see [DL15].

Figure 2.2 lists important properties of all these guarded fragments, using ✓ and ✗ to denote positive and negative answers, respectively. In the satisfiability checking row in Figure 2.2, all guarded fragments are assumed to have a fixed signature, so that they satisfy the assumptions of the thesis.

As UNF is a trivial special case of GNF with $x \approx x$ as guards, this thesis does not independently discuss UNF. The decision procedures established for querying in GNF instantly are the practical decision procedures for querying in UNF.

The first-order guarded fragments

Guarded quantification fragments We now formally define the *guarded*, the *loosely guarded* and the *clique guarded fragments*. In guarded quantification fragments constants are freely allowed, but not equality.

Definition 1. *The* guarded fragment (GF) *is a fragment of FOL without function symbols, 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 connectives.*
4. *Let F be a guarded formula and G an atom. Then $\exists\overline{x}(G \wedge F)$ and $\forall\overline{x}(G \rightarrow F)$ belong to GF if all free variables of F occur in G.*

Definition 2. *The* loosely guarded fragment (LGF) *is a fragment of FOL without function symbols, inductively defined 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 connectives.*
4. *Let F be a loosely guarded formula and $\mathbb{G}$ a conjunction of atoms. Then $\forall\overline{x}(\mathbb{G} \rightarrow F)$ and $\exists\overline{x}(\mathbb{G} \wedge F)$ belong to LGF if*
   (a) *all free variables of F occur in $\mathbb{G}$, and*
   (b) *for each variable x in $\overline{x}$ and each variable y occurring in $\mathbb{G}$ that is distinct from x, x and y co-occur in an atom of $\mathbb{G}$.*

Definition 3. *The* clique guarded fragment (CGF) *is a fragment of FOL without function symbols, inductively defined as follows:*

1. $\top$ *and* $\bot$ *belong to CGF.*
2. *If A is an atom, then A belongs to CGF.*

3. *CGF is closed under Boolean connectives.*
4. *Let F be a clique guarded formula and $\mathbb{G}(\overline{x}, \overline{y})$ a conjunction of atoms. Then $\forall \overline{z}(\exists \overline{x}\mathbb{G}(\overline{x}, \overline{y}) \rightarrow F)$ and $\exists \overline{z}(\exists \overline{x}\mathbb{G}(\overline{x}, \overline{y}) \wedge F)$ belong to CGF, if*
   (a) *all free variables of F occur in $\overline{y}$, and*
   (b) *each variable in $\overline{x}$ occurs in only one atom of $\mathbb{G}(\overline{x}, \overline{y})$, and*
   (c) *for each variable z in $\overline{z}$ and each variable y occurring in $\mathbb{G}(\overline{x}, \overline{y})$ that is distinct from z, z and y co-occur in an atom of $\exists \overline{x}\mathbb{G}(\overline{x}, \overline{y})$.*

In *4.* of **Definitions 1–3**, the atom G, the conjunction of atoms $\mathbb{G}$ and the formula $\exists \overline{x}\mathbb{G}(\overline{x}, \overline{y})$ are called the *guard*, the *loose guard* and the *clique guard* for the formula F, respectively.

In guarded formulas all quantified formulas contain at least one guard. Consider the following formulas.

$$F_1 = A(x) \qquad F_2 = \forall x[A(x)] \qquad F_3 = \forall x[A(x, y) \rightarrow B(x, y)]$$
$$F_4 = \forall x[A(x, y) \rightarrow \exists y(B(y, z))] \qquad F_5 = \forall x[A(x, y) \rightarrow \perp]$$
$$F_6 = \exists x[A(x, y) \wedge \forall z(B(x, z) \rightarrow \exists u(R(z, u)))]$$
$$F_7 = \forall x[P(x) \rightarrow \exists y(R(x, y) \wedge \forall z(R(y, z) \rightarrow P(z))))]$$

The formulas F_1, F_3, F_5, F_6 and F_7 are guarded formulas, but the rest are not. The formulas F_2 and F_4 are not guarded as they do not contain a 'guard' atom. The formula F_7 is the standard translation [BRV01b] of the description logic $\mathcal{ALCHOI}$ axiom $P \sqsubseteq \exists R.\forall R.P$ and the modal formula $P \rightarrow \Diamond \Box P$.

By the standard translation, description logic $\mathcal{ALCHOI}$ axioms and modal formulas can be translated into non-guarded formulas. For example applying the standard translation to $\forall R.P \sqsubseteq \perp$ produces

$$F = \forall x(\forall y(R(x, y) \rightarrow P(y)) \rightarrow \perp)$$

with x not guarded. Nonetheless as these translated formulas contain only one 'unguarded' variable x, this variable can be regarded as being implicitly guarded by $x \approx x$. In this example F can be reformulated as a logical equivalent formula $F' = \forall x(x \approx x \rightarrow \forall y(R(x, y) \rightarrow P(y)) \rightarrow \perp))$. The formula F' is a guarded formula with equality, handled by the decision procedure for querying in GNF as GNF subsumes GF with equality.

LGF strictly extends GF by allowing a conjunction of atoms to pairwise guard quantified variables. For example $\forall z((R(x,z) \wedge R(z,y)) \rightarrow P(z))$ is loosely guarded, not guarded. The standard translation the temporal formula P `until` Q, namely

$$\exists y(R(x,y) \wedge Q(y) \wedge \forall z((R(x,z) \wedge R(z,y)) \rightarrow P(z)))),$$

belongs to LGF, but not GF. The transitivity formula

$$\forall xyz((R(x,y) \wedge R(y,z)) \rightarrow R(x,z))$$

is neither guarded nor loosely guarded.

CGF further extends LGF by allowing existential quantification to atoms in loose guards. For example,

$$F = \forall x_1 x_2(G(x_1,x_2) \rightarrow \forall x_3(A_1(x_1,x_3) \wedge B_1(x_2,x_3) \rightarrow \exists x_6 D(x_1,x_6))),$$

is a loosely guarded formula, in which $A_1(x_1,x_3) \wedge B_1(x_2,x_3)$ and $G(x_1,x_2)$ are, respectively, the loose guards for $\forall x_3(A_1(x_1,x_3) \wedge B_1(x_2,x_3) \rightarrow \exists x_6 D(x_1,x_6))$ and F. By adding existential quantifications the loose guards of $\forall x_3(A_1(x_1,x_3) \wedge B_1(x_2,x_3) \rightarrow \exists x_6 D(x_1,x_6))$, one obtains the clique guarded formula

$$\forall x_1 x_2(G(x_1,x_2) \rightarrow \forall x_3(\exists x_4 x_5(A(x_1,x_3,x_4) \wedge B(x_2,x_3,x_5)) \rightarrow \exists x_6 D(x_1,x_6))),$$

where $\exists x_4 x_5(A(x_1,x_3,x_4) \wedge B(x_2,x_3,x_5))$ is the clique guard for

$$\forall x_3(\exists x_4 x_5(A(x_1,x_3,x_4) \wedge B(x_2,x_3,x_5)) \rightarrow \exists x_6 D(x_1,x_6))),$$

and $G(x_1,x_2)$ is the guard for the entire formula.

Although equality is prohibited in the *guarded quantification fragments*, it is allowed in the *guarded negation fragments*, which in general subsume the guarded quantification fragments.

Guarded negation fragments Next, we formally define the *unary negation*, the *guarded negation* and the *clique guarded negation fragments*. Compared to the guarded quantification fragments, both constants and equality are freely allowed in guarded negation fragments.

Definition 4. *The* unary negation fragment (UNF) *is a fragment of* FOL$_{\approx}$ *without functional symbols, inductively defined as follows:*

1. *$\top$ and $\bot$ belong to* UNF.
2. *If A is an atom, then A belongs to* UNF.
3. *If A and B are atoms, then $A \vee B$ and $A \wedge B$ belong to* UNF.
4. *If F belongs to* UNF, *then $\exists \overline{x} F$ belongs to* UNF.
5. *Let F be a unary negation formula. Then $\neg F$ belongs to* UNF *if F contains only one free variable.*

Definition 5. *The* guarded negation fragment (GNF) *is a fragment of* FOL$_{\approx}$ *without functional symbols, inductively defined as follows:*

1. *$\top$ and $\bot$ belong to* GNF.
2. *If A is an atom, then A belongs to* GNF.
3. *If A and B are atoms, then $A \vee B$ and $A \wedge B$ belong to* GNF.
4. *If F belongs to* GNF, *then $\exists \overline{x} F$ belongs to* GNF.
5. *Let F be a guarded negation formula and G an atom. Then $G \wedge \neg F$ belongs to* GNF *if all free variables of F belong to the variables of G.*

Definition 6. *The* clique guarded negation fragment (CGNF) *is a fragment of* FOL$_{\approx}$ *without functional symbols, inductively defined as follows:*

1. *$\top$ and $\bot$ belong to* CGNF.
2. *If A is an atom, then A belongs to* CGNF.
3. *If A and B are atoms, then $A \vee B$ and $A \wedge B$ belong to* CGNF.
4. *If F belongs to* CGNF, *then $\exists \overline{x} F$ belongs to* CGNF.
5. *Let F be a clique guarded negation formula and $\mathbb{G}(\overline{x}, \overline{y})$ a conjunction of atoms. Let $\overline{z}$ denote the free variables of F. Then $\exists \overline{x} \mathbb{G}(\overline{x}, \overline{y}) \wedge \neg F$ belongs to* CGNF *if*
   (a) *$\overline{z}$ is a subset of $\overline{y}$, and*
   (b) *each variable in $\overline{x}$ occurs in only one atom of $\mathbb{G}(\overline{x}, \overline{y})$, and*
   (c) *each pair of distinct variables in $\overline{y}$ co-occurs in an atom of $\exists \overline{x} \mathbb{G}(\overline{x}, \overline{y})$.*

In *5.* of **Definitions 5–6**, the atom G and the formula $\exists \overline{x} \mathbb{G}(\overline{x}, \overline{y})$ are called the *guard* and the *clique guard* for the formula F, respectively.

GNF subsumes GF since every *guarded sentence* is expressible in GNF, but not vice-versa [BtCS15, Proposition 2.2]. However not all *guarded formulas* can be transformed to a *guarded negation formula*; consider $\neg A(x, y, z)$. By limiting the guard of a *guarded negation formula* to be an equality literal, one obtains a *unary negation formula*.

Comparing *5c.* in the definitions of GCNF and CGF, an important distinction is that the *pairwise guarded condition* is changed from the quantified variables to the variables occurring in the negated formulas. In the clique guarded formula $\exists\overline{z}(\exists\overline{x}\mathbb{G}(\overline{x}, \overline{y}) \wedge F)$ the *pairwise guardedness* is required for the variables in $\overline{z}$ and the variables in $\overline{y}$, whereas in the clique guarded negation formula $\exists\overline{x}\mathbb{G}(\overline{x}, \overline{y}) \wedge \neg F$ the *pairwise guardedness* is imposed on the variables in $\overline{y}$. A sample clique guarded negation formula is:

$$F = \left[\begin{array}{lll} \neg\exists x_1 x_2 x_3(& \exists y_1 y_2(A_1(x_1, x_2, y_1) \wedge A_1(x_2, x_3, y_2) \wedge x_1 \approx x_3) \wedge & \\ & \neg\exists x_4(B(x_1, x_2, x_4) \wedge B(x_2, x_3, x_4)) &) \end{array}\right].$$

Using the notion of *generalised guard* for GNF in [BtCO12], we obtain *generalised guards*, *generalised loose guards* and *generalised clique guards*, by the following method. Suppose F is a formula in any of guarded first-order fragments and F_1 is a disjunction of existentially quantified atoms such that the free variables of F occur in each atom of F_1. Then if one adds a *guard*, a *loose guard* or a *clique guard* to F, any atom A in these guards can be replaced by F_1, forming a *generalised guard*, a *generalised loose guard* or a *generalised clique guard* for F, respectively. By replacing A with F_1 in F, one obtains the *generalised formula* F'. The formula F' extends the expressive power of F if F belongs to GF, LGF or CGF, otherwise F and F' are of the same expressivity. For example by replacing $A(x, y)$ by $\exists x_1 A_1(x, y, x_1) \vee \exists x_2 A_2(x, y, x_2)$ for the guarded formula

$$F = \forall x(A(x, y) \rightarrow \exists y B(y)),$$

one obtains the *generalised guarded formula*

$$F' = \forall x((\exists x_1 A_1(x, y, x_1) \vee \exists x_2 A_2(x, y, x_2)) \rightarrow \exists z B(y, z)).$$

The formula F' is not in GF or LGF due to the occurrence of the existential quantifiers and the disjunction in its guard, and it is not in CGF as the *generalised guard* $\exists x_1 A_1(x, y, x_1) \vee \exists x_2 A_2(x, y, x_2)$ is a disjunction, not conjunction.

2.2 The BCQ answering and rewriting problems

The queries considered in this thesis are *unions of Boolean conjunctive queries*. A *conjunctive query* (CQ) is a first-order formula (with equality) of the form $\exists \overline{x} F(\overline{x}, \overline{y})$, where $F(\overline{x}, \overline{y})$ is a conjunction of atoms, with only variables and constants occurring as arguments. A *Boolean conjunctive query* (BCQ) is a first-order sentence (with equality) of the form $\exists \overline{x} F(\overline{x})$ where $F(\overline{x})$ is a conjunction of atoms with only variables and constants occurring as arguments. A *Boolean conjunctive query with equality* ($\mathsf{BCQ}_{\approx}$) is a BCQ with equality literals allowed. *A union of Boolean conjunctive queries* (*union of BCQs*) is a disjunction of BCQs (and a union of $\mathsf{BCQ}_{\approx}$s).

Recall that equality is allowed in the *guarded negation fragments*, but not in the *guarded quantification fragments*. Consequently in querying for the guarded negation fragments, we consider $\mathsf{BCQ}_{\approx}$s as the query language, and for the rest of the query tasks we consider BCQs. For readability BCQ and $\mathsf{BCQ}_{\approx}$ are mostly not distinguished in the rest of the thesis.

BCQ answering problems

Now we give the formal definition of the BCQ answering problem we investigate.

Problem 1. *Given a set* Σ *of first-order formulas (with equality), a set* D *of ground atoms and a union* q *of BCQs, can a saturation-based procedure decide* $\Sigma \cup \mathrm{D} \models q$*?*

Since ground atoms belong to any of the considered guarded fragments, **Problem 1** can be refined as follows.

Problem 2. *Given a set* Σ *of first-order formulas (with equality) and a union* q *of BCQs, can a saturation-based procedure decide* $\Sigma \models q$*?*

In the formal definition of the BCQ answering problem for the guarded first-order fragments, the formulation of **Problem 2** is used for its simplicity.

In **Problem 2** one negates the given union of BCQs, obtaining the *negated BCQ*, so that **Problem 2** is reduced to deciding whether the combination of the given formulas and the negated BCQs is satisfiable. BCQs (a union of BCQs) and their negations are expressible in the *guarded negation fragments*, but not in the *guarded quantification fragments*. Figure 2.3 summaries the relationship of the aforementioned guarded fragments, the negated BCQ and FOL.

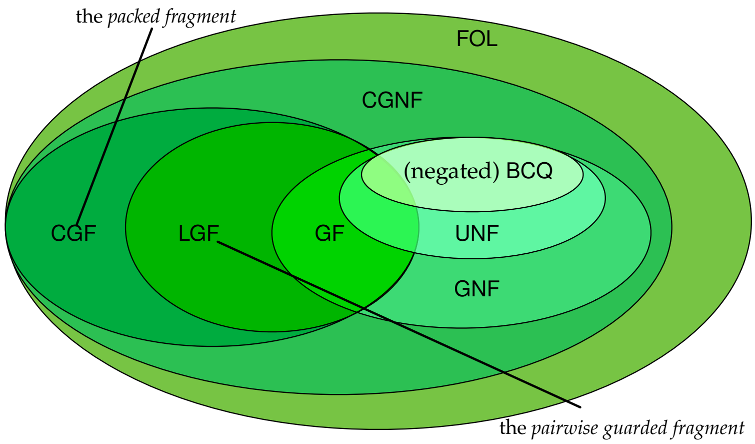

Figure 2.3: The relationship between the considered fragments, negated BCQ and FOL

The computational complexity of the BCQ answering problem for GF is 2ExpTime-complete [BGO14]. By the fact that formulas in CGF and the negated BCQs are expressible in CGNF, the problem of answering BCQs for LGF and/or CGF is a subproblem of deciding satisfiability of CGNF. Therefore, answering BCQs for LGF and/or CGF is 2ExpTime-complete [BtCS15]. Due to the fact that the negated BCQs are expressible in GNF or CGNF, the problems of BCQ answering for GNF and/or CGNF have the same complexity as the satisfiability checking problem for GNF and CGNF, which is 2ExpTime-complete [BtCS15]. These complexity results mean that the problems of answering BCQs for GF, LGF, CGF, GNF and/or CGNF are all decidable.

Saturation-based BCQ rewriting and back-translation problems

The saturation-based rewriting problem is motivated by the *first-order rewritability*, introduced for the lightweight description logic DL-Lite family, tackling the *ontology-mediated querying* tasks [CGL+07]. For a union of BCQs, first-order rewritability is formally defined as follows.

Definition 7. *Given a set Σ of first-order formulas (with equality), a set* D *of ground atoms and a union q of BCQs, q and Σ are said to be* first-order rewritable *if q and Σ can be complied into a (function-free) first-order formula Σ_q such that for any* D*, $\Sigma \cup \mathrm{D} \models q$ if and only if $\mathrm{D} \models \Sigma_q$.*

As given in **Definition 7**, the *first-order rewritability* is devised such that the entailment checking problem $\Sigma \cup \mathrm{D} \models q$ is reduced to the model checking

	GF	LGF	CGF	UNF	GNF	CGNF
BCQ answering	2ExpTime [BGO14]	2ExpTime [BtCS15]		2ExpTime [tCS13]	2ExpTime [BtCS15]	
First-order rewritability	✗ [BBLP18, BBGP21]			open	✗ [BBLP18, BBGP21]	

Figure 2.4: Known properties of querying in the studied fragments

problem $\mathrm{D} \models \Sigma_q$. The latter is in the AC^0 complexity class [Var95]. Though desirable as the first-order rewritability is, BCQs (and their extensions thereof) and GF (and its extensions thereof) do not have this property; see [BBGP21, Example 2.2] and [BBLP18, Example 1]. Figure 2.4 summarises the known results for the complexity of BCQ answering and first-order rewritability of all the targeted guarded fragments (with respect to BCQs). In the figure, the ✗ mark means a negative answer.

Proposing a new perspective to the rewriting problem, we consider it as a back-translation problem, formally stated as follows.

Problem 3. *Given a set Σ of first-order formulas (with equality), a set* D *of ground atoms and a union q of* BCQ*s, can we compute a (function-free) first-order formula (with equality) Σ_q that is the negated back-translation of the saturated clausal set of $\Sigma \cup \{\neg q\}$ such that $\Sigma \cup \mathrm{D} \models q$ if and only if $\mathrm{D} \models \Sigma_q$?*

Problem 3 is formalised in a way so that it is established on the solutions to **Problem** 1. In **Problem** 1 the problem of $\Sigma \cup \mathrm{D} \models q$ is generally considered as that of checking unsatisfiability of $\{\neg q\} \cup \Sigma \cup \mathrm{D}$. Without D, from $\{\neg q\} \cup \Sigma$ one can either derive $\perp$ or a saturated clausal set N. Suppose $\perp$ is derived. This case is trivial when $\Sigma \models q$ and hence Σ_q in **Problem** 3 is $\top$. Otherwise N is derived, and **Problem** 3 then aims to back-translate N to a first-order formula, which is then negated and used as Σ_q in deciding $\mathrm{D} \models \Sigma_q$.

The technical challenge in **Problem** 3 is the back-translation of a clausal set to a first-order formula, which is a form of *second-order quantifier elimination* [GSS08a]. In **Problem** 3 it is the existentially quantified Skolem function symbols and constants are intended to be eliminated.

Chapter 3

Saturation-based theorem proving for first-order logic

This chapter is organised as follows. **Section 3.1** introduces basic notions of first-order logic and first-order clausal logic. **Section 3.2** and **Section 3.3** give the clausification and the back-translation techniques, respectively. **Section 3.4** presents fundamentals for saturation-based inference systems.

3.1 First-order logic

Basic notions in first-order logic

This section formally defines the syntax of first-order logic (FOL). Let C, F, P and V be four countably infinite sets that are pair-wise disjoint. The elements in C, F and P are the *constant symbols* (*constants*), the *function symbols* and the *predicate symbols*. We say a tuple (C, F, P) is a *signature*. The elements in V are *variables*. A function symbol or a predicate symbol is considered with a unique integer, denoting the *arity* of that symbol. A predicate symbol of arity zero is a *propositional variable*. Note that in this thesis the function symbols are considered as non-constant function symbols.

A *term* is either a constant, or a variable, or $f(t_1, \ldots, t_n)$ if i) f is a function symbol of arity n and ii) $t_1, \ldots, t_n$ are terms. A term s is a *subterm* of a term t if s is identical to t, or $t = f(t_1, ..., t_n)$ and s is a subterm of one of terms $t_1, ..., t_n$. A term s is a *strict subterm* of a term t if s is a subterm of t, and s is not identical to t. A *compound term* is a term that is neither a constant nor a variable.

We use the following logical connectives: $\top$ (verum), $\bot$ (falsum), $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), $\rightarrow$ (implication) and $\leftrightarrow$ (double implication). A *Boolean connective* is one of the following symbols: $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$. The symbol $\forall$ is the *universal quantifier* and is read 'for all'. The symbol $\exists$ is the *existential quantifier* and is read 'there exists'.

If P is a predicate symbol of arity n, and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is an *atomic formula* (*atom*). We regard $\top$ and $\bot$ as atoms. A *literal* is either an atom (denoted as a *positive literal*), or a negated atom (denoted as a *negative literal*). The literal L denotes either an atom A or a negated atom $\neg A$. A *literal in propositional logic* is either a propositional variable or its negation. Two literals A and $\neg A$ are called a *complementary literals*. For a literal $L(t_1, \ldots, t_n)$ and a compound term $f(t_1, \ldots, t_n)$, i) a term in $t_1, \ldots, t_n$ is called an *argument* of L and t, respectively, and ii) $t_1, \ldots, t_n$ is called the *argument list* of L and t, respectively.

A set of *first-order formulas* (*formulas*) over a signature $(\mathtt{C}, \mathtt{F}, \mathtt{P})$ is inductively defined as follows.

1. If A is an atom, then A and $\neg A$ are first-order formulas.
2. First-order formulas are closed under Boolean connectives.
3. If F is a first-order formula and x is a variable, then $\forall x F$ and $\exists x F$ are first-order formulas.

The *proper subformula* of a first-order formula F is inductively defined as follows.

1. Atomic formulas have no proper subformulas.
2. $F = \neg F_1$: The proper subformulas of F are F_1 and all proper subformulas of F_1.
3. $F = F_1 \circ F_2$ where $\circ$ denotes a Boolean connective: The proper subformulas of F are F_1, F_2, and all proper subformulas of F_1 and F_2.
4. $F = QxF_1$ where Q denotes a quantifier: The proper subformulas of F are F_1 and all proper subformulas of F_1.

The *subformula* of F are F and the proper subformulas of F. The *immediate subformula* of a first-order formula F is inductively defined as follows.

1. Atomic formulas have no immediate subformulas.
2. $F = \neg F_1$: The immediate subformula of F is F_1.
3. $F = F_1 \circ F_2$ where $\circ$ denotes a Boolean connective: The immediate subformulas of F are F_1 and F_2.

4. $F = QxF_1$ where Q denotes a quantifier: The immediate subformula of F is F_1.

In a quantified formula $\forall xF$, x is the *quantified variable* and F is the *scope* of the quantified variable x. An occurrence of a variable x in a first-order formula F is a *free variable* of F if and only if x is not within the scope of quantified variables. A variable is a *bound variable* of F if it is not a free variable of F. A *sentence* (*closed formula*) is a first-order formula without free variables.

If a signature ($\mathtt{C}$, $\mathtt{F}$, $\mathtt{P}$) allows special predicate symbols $\approx$ and $\not\approx$, then we consider *first-order logic with equality*, an extension of FOL. We use the *infix notation* for equational atoms, denoted as $s \approx t$. We use the notation $s \not\approx t$ to denote the negation of $s \approx t$. The literals $s \approx t$ and $s \not\approx t$ are called an *equality literal* and an *inequality literal*, respectively.

First-order clauses

A *first-order clause* (*clause*) is a multiset of literals, denoting a finite disjunction of literals. A *first-order clause with equality* (*clause with equality*) is a first-order clause that may contain the predicate symbols $\approx$ and $\not\approx$. A *subclause* D of a clause C, is a sub-multiset D of C. A *set S of clause* (*clausal set S*) is a conjunction of all clauses in S, where every variable in S is considered to be universally quantified.

An *expression* E is either a term, or an atom, or a literal or a clause. An expression E is a *subexpression* of an expression E_1 if E occurs in E_1. An expression E is a *proper subexpression* of an expression E_1 if E is a subexpression of E_1 and E is not identical to E_1. The expressions E_1 and E_2 are *variable-disjoint* if they share no common variables. A *ground expression* is a variable-free expression. A clause C is *Horn* if C contains at most one positive literal. A clause C is *negative* if C contains only negative literals. A clause C is *positive* if C contains only positive literals. A clause C is *decomposable* if C can be partitioned into two variable-disjoint subclauses, or else C is *indecomposable*.

Customised definitions

Now we give definitions particularly devised for this thesis.

The set of variables that occurs in an expression E is denoted as $\mathrm{var}(E)$. We use notations $C(t)$ and $F(t)$ to, respectively, denote a clause with equality C and a formula with equality F, in which the term t occurs.

To describe argument positions in a pair of terms, in [dNdR03] the notion *pair* is introduced. Given two expressions $E_1 = A(\ldots, t, \ldots)$ and $E_2 = B(\ldots, s, \ldots)$, we say t *pairs* s (with respect to t of E_1 and s of E_2) if the argument position of t in A is the same as that of u in B. For example in $A(x_1, f(x_1, x_2), x_2)$ and $B(g(y_1), y_1, y_2)$, x_1 pairs $g(y_1)$, $f(x_1, x_2)$ pairs y_1, and x_2 pairs y_2.

The *depth of a term* t is denoted as $\text{dep}(t)$, defined as follows:

1. If t is a variable or a constant, then $\text{dep}(t) = 0$, and
2. if t is a compound term $f(t_1, \ldots, t_n)$, then $\text{dep}(t) = 1 + max(\{\text{dep}(t_i) \mid 1 \leq i \leq n\})$.

The *depth of an expression* E is the deepest term depth in E, denoted as $\text{dep}(E)$. If no terms occur in an expression E, then $\text{dep}(E) = 0$. The *width of the expression* E is the number of distinct variables in E. For example, given the clause $C = \neg A_1(f(x_1), x_1) \vee A_2(x_1, x_2) \vee A_3(x_2, g(x_2, x_3))$, the depth of C is one since the deepest term in C is $f(x_1)$, and $g(x_2, x_3)$ and the width of C is three since C contains three distinct variables x_1, x_2 and x_3.

In [FLTZ93] the notion *covering* for terms is introduced. In this thesis, we generalise this notion so that it is applicable to clauses. A term t is *covering* if for every compound subterm s of t, the variables sets of s and t are identical, namely $\text{var}(s) = \text{var}(t)$. A literal L is *covering* if each argument of L is either a constant, or a variable or a covering term t satisfying $\text{var}(t) = \text{var}(L)$. A clause C is *covering* if for each literal L in C, each argument of L is either a constant, or a variable, or a covering term t satisfying $\text{var}(t) = \text{var}(C)$. For instance, $C_1 = A_1(f(x_1, x_2, a), x_1) \vee A_2(x_1, x_2)$ is a covering clause since the only compound term $f(x_1, x_2, a)$ in C_1 satisfies that $\text{var}(f(x_1, x_2, a)) = \text{var}(C_1)$. The clause $C_2 = A_1(f(x_1), x_1) \vee A_2(g(x_2))$ is not covering since $\text{var}(C) \neq \text{var}(g(x_2))$, however $A_1(f(x_1), x_1)$ is a covering literal since $\text{var}(f(x_1)) = \text{var}(A_1(f(x_1), x_1))$.

The notions of *flatness* and *simpleness* are introduced in [GdN99]. We use *flat* and *simple* to define an expression that is of depth zero and of depth zero or one, respectively. A compound term $f(t_1, \ldots, t_n)$ is *flat* if each term in $t_1, \ldots, t_n$ is either a variable or a constant. A literal L is *flat* if each argument in L is either a constant or a variable. A clause C is *flat* if all literals in C are flat. A literal L is *simple* if each argument of L is either a variable, or a constant or a flat compound term. A clause C is *simple* if all literals in C are simple. A literal (clause) is a *compound-term literal* (*compound-term clause*) if the depth of this literal (clause)

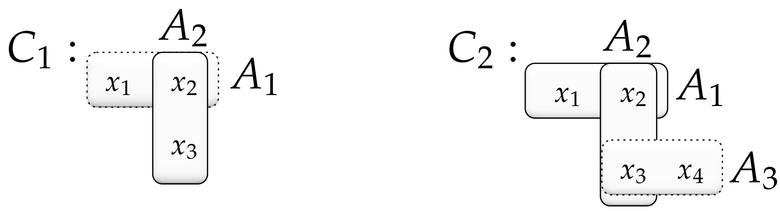

Figure 3.1: The hypergraphs associated with C_1 and C_2

is one. The definitions of the compound-term literal (clause) are restricted to non-nested compound terms since this thesis only focuses on simple clauses. For example $A_1(x_1)$, $\neg A_1(x_1, a_1) \vee A_2(x_2, x_3)$ and $A_1(a_1) \vee A_2(a_2, a_3)$ are flat and simple clauses, however the clauses $\neg A_1(f(x_1), a_1) \vee A_2(x_2, x_3)$ and $A_1(a_1) \vee A_2(f(a_2), a_3)$ are simple but not flat as they contain flat compound terms. In fact they are compound-term clauses. The clause $\neg A_1(x_1, a_1) \vee A_2(x_2, f(f(x_3)))$ is neither flat nor simple as it contains a non-flat compound term $f(f(x_3))$.

An expression is *flat* if each of its argument is either a variable or a constant. Given a flat expression E and a term t occurring in E, we use $\mathrm{Occ}(t, E)$ to denote the number of occurrences of t in E. For example, $\mathrm{Occ}(x, f(x, y, x)) = 2$ as x occurs twice in $f(x, y, x)$ and $\mathrm{Occ}(a, \neg A_1(x, y) \vee A_2(z, a)) = 1$ as a occurs once in $\neg A_1(x, y) \vee A_2(z, a)$.

Suppose C is a flat clause and $\mathcal{H}(V, E)$ is a hypergraph consisting of a set V of vertices and a set E of hyperedges. Then we *associate the hypergraph* $\mathcal{H}(V, E)$ *with* C as follows: The set V of vertices consists of all variables in C, and the set E of hyperedges contains, for each literal L in C, the set of variables that appear in L. To represent a flat clause by a hypergraph, we use rectangles and variable symbols to represent hyperedges and vertices, respectively. Dotted-line and solid-line rectangles represent positive and negative literals, respectively, and negation symbols are omitted. Figure 3.1 presents the hypergraphs associated with $C_1 = A_1(x_1, x_2) \vee \neg A_2(x_2, x_3)$ and $C_2 = \neg A_1(x_1, x_2) \vee \neg A_2(x_2, x_3) \vee A_3(x_3, x_4)$.

In the rest of the thesis, we use the following notational conventions:

- $x, y, z, u, v, x_1, \ldots$ for variables
- $f, g, h, \ldots$ for function symbols
- $p, p_1, \ldots$ for propositional variables
- $C, D, Q, C_1, \ldots$ for clauses
- $L, L_1, \ldots$ for literals
- $a, b, c, a_1, \ldots$ for constant symbols
- $A, B, G, P, \ldots$ for predicate symbols
- $F, F_1, \ldots$ for formulas
- $s, t, u, \ldots$ for terms

3.2 Clausification techniques

This section gives the techniques that transform a formula to a clausal set. This transformation is called the *clausal normal form transformation* or *clausification*.

Negation normal form

A formula F is in *negation normal form* if every negation symbol in F occurs directly in front of an atom. Exhaustively applying the following rules to a formula transforms it to negation normal form.

The NNF rules

$$
\begin{array}{rcl@{\qquad}rcl}
F_1 \leftrightarrow F_2 & \Rightarrow & (F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1) & & & \\
\neg(F_1 \vee F_2) & \Rightarrow & \neg F_1 \wedge \neg F_2 & \neg(F_1 \wedge F_2) & \Rightarrow & \neg F_1 \vee \neg F_2 \\
\neg \forall x F & \Rightarrow & \exists x \neg F & \neg \exists x F & \Rightarrow & \forall x \neg F \\
F_1 \rightarrow F_2 & \Rightarrow & \neg F_1 \vee F_2 & \neg\neg F & \Rightarrow & F \\
\neg \top & \Rightarrow & \bot & \neg \bot & \Rightarrow & \top
\end{array}
$$

Miniscoping and prenex normal form

A formula F is in *prenex normal form* if $F = Q_1 x_1 \ldots Q_n x_n F_1$ where $Q_1, \ldots, Q_n$ are quantifiers and F_1 is a quantifier-free first-order formula. In contrast to prenex normal form, a formula F is *anti-prenex normal form* if the quantifiers of F are moved to its quantified variables as much as possible.

Quantifiers are moved to its quantified variables using

The Miniscoping rules

$$
\begin{array}{rcll}
\exists x(F_1 \vee F_2) & \Rightarrow & \exists x F_1 \vee F_2 & \text{if } x \text{ does not occur in } F_2. \\
\exists x(F_1 \wedge F_2) & \Rightarrow & \exists x F_1 \wedge F_2 & \text{if } x \text{ does not occur in } F_2. \\
\forall x(F_1 \vee F_2) & \Rightarrow & \forall x F_1 \vee F_2 & \text{if } x \text{ does not occur in } F_2. \\
\forall x(F_1 \wedge F_2) & \Rightarrow & \forall x F_1 \wedge F_2 & \text{if } x \text{ does not occur in } F_2. \\
\forall x(F_1 \wedge F_2) & \Rightarrow & \forall x F_1 \wedge \forall x F_2 & \text{if } x \text{ occurs in both } F_1 \text{ and } F_2. \\
\exists x(F_1 \vee F_2) & \Rightarrow & \exists x F_1 \vee \exists x F_2 & \text{if } x \text{ occurs in both } F_1 \text{ and } F_2.
\end{array}
$$

Structural transformation

Let F_1 be a subformula of a formula F. Then F_1 has *positive polarity (with respect to F)* if and only if F_1 occurs in the scope of an even number of implicit or explicit negation symbols. F_1 has *negation polarity (with respect to F)* if and only if F_1 occurs in the scope of an odd number of implicit or explicit negation symbols.

A formula is renamed using

The Trans rules

$F[F_1(x)] \Rightarrow F[P(x)] \land \forall x(P(x) \rightarrow F_1(x))$
if F_1 has the positive polarity and P is a predicate symbol that does not occur in F.

$F[F_1(x)] \Rightarrow F[P(x)] \land \forall x(F_1(x) \rightarrow P(x))$
if F_1 has the negative polarity and P is a predicate symbol that does not occur in F.

The **Trans** rules is also referred to as the *formula renaming* technique. The *formula renaming technique* is also applicable to clauses, which can be seen as a sentence with all variables universally quantified. Such application can be implemented in a more general approach so that the polarity of the introduced literals is not restricted. Consider a clause $C = C_1 \lor C_2$. By introducing a fresh predicate symbol P with $\mathrm{var}(P) = \mathrm{var}(C_2)$ for C_2, C can be renamed as either $\{C_1 \lor P, \neg P \lor C_2\}$ or $\{C_1 \lor \neg P, P \lor C_2\}$.

Skolemisation

Skolemisation aims to eliminate existential quantifications and existentially quantified variables from a formula.

A formula is skolemised using

The Skolem rule

$$\frac{\forall x_1 \ldots \forall x_n \exists y F(y)}{\forall x_1 \ldots \forall x_n F(f(x_1, \ldots, x_n))}$$

if f is a Skolem function symbol that does not occur in $\forall x_1 \ldots \forall x_n \exists y F(y)$.

In the **Skolem** rule, we say f is a *Skolem function symbol*, and it is a *Skolem constant symbol* (*Skolem constant*) if $n = 1$. The term $f(x_1, \ldots, x_n)$ is a *Skolem compound term*. A *Skolem term* is either a Skolem compound term or a Skolem constant. For more advanced and comprehensive Skolemisation techniques such as *Strong Skolemisation* and *Optimised Skolemisation*, see [NW01, Section 5]. In the thesis the **Skolem** rule is sufficient for the hope-for results.

Conjunctive normal form

A formula $F = F_1 \vee \ldots \vee F_n$ is a *disjunction* or *disjunctive formula* and each F_i is a *disjunct* of F. A formula $F = F_1 \wedge \ldots \wedge F_n$ is a *conjunction* or *conjunctive formula* and each F_i is a *conjunct* of F. A formula F is in *conjunctive normal form* if and only if F is a conjunction of disjunctions of literals, and F is in *disjunctive normal form* if and only if F is a disjunction of conjunctions of literals.

A formula is transformed to conjunctive normal form using

The CNF rules

$$\begin{array}{lcll} \forall x F_1 \vee F_2 & \Rightarrow & \forall x(F_1 \vee F_2) & \text{if } x \text{ does not occur in } F_2. \\ \forall x F_1 \wedge F_2 & \Rightarrow & \forall x(F_1 \wedge F_2) & \text{if } x \text{ does not occur in } F_2. \\ F \vee (F_1 \wedge F_2) & \Rightarrow & (F \vee F_1) \wedge (F \vee F_2) & \end{array}$$

By the **CNF** rules, transforming a formula to conjunctive normal form can cause exponential blow-up due to the following distribution of disjunctions.

$$F = (F_1^1 \wedge F_2^1 \wedge \ldots \wedge F_m^1) \vee (F_1^2 \wedge F_2^2 \wedge \ldots \wedge F_m^2) \vee \ldots \vee (F_1^n \wedge F_2^n \wedge \ldots \wedge F_m^n).$$

A commonly used method to reduce the above blow-up to a polynomial-time problem is applying the **Trans** rule to F, that is, introducing a fresh predicate symbols for each conjunction that is under a disjunction in F. We introduce fresh predicate symbols P_i for $(F_1^i \wedge F_2^i \wedge \ldots \wedge F_m^i)$ for all i with $1 \leq i \leq n$. Then F is transformed into

$$\begin{array}{l} P_1 \rightarrow F_1^1 \wedge F_2^1 \wedge \ldots \wedge F_m^1, \quad , \ldots, \quad P_n \rightarrow F_1^n \wedge F_2^n \wedge \ldots \wedge F_m^n, \\ P_1 \vee \ldots \vee P_n, \end{array}$$

which can be transformed to conjunctive normal form in polynomial time.

The above rules are standard clausification techniques in [NW01], transforming a first-order formula to clausal normal forms.

Lemma 3.1 ([NW01])**.** *The **NNF**, the **Miniscoping** and the **CNF** rules preserve logical equivalence. The **Trans** rules and the **Skolem** rule preserve satisfiability.*

3.3 Back-translation techniques

Pre-conditions for a successful back-translation

In [Eng96, Chapter 5], it is shown that a clausal set N can be unskolemised if N is *normal*, *unique*, *globally linear* and *globally consistent*. To avoid ambiguity we use the word *compatible* to replace the word *consistent*.

Now we formally introduce these definitions.

Definition 8. *A compound term t* is compatible with *another distinct compound term s if the argument lists of t and s are identical. A clause C is* compatible *if in C, compound terms that are under the same function symbol are compatible.*

A clausal set N is locally compatible *if all clauses in N are compatible. A clausal set N is* globally compatible *if in N, compound terms that are under the same function symbol are compatible.*

Definition 9. *Compound terms t and s are* linear *if the set of arguments of t is a subset of that of s or vice-versa. A clause C is* linear *if each pair of compound terms in C is linear.*

A clausal set N is locally linear *if every clause in N is linear. A clausal set N is* globally linear *if each pair of compound terms in N is linear.*

Definition 10. *A compound term $f(t_1, \ldots, t_n)$ is* normal *if $t_1, \ldots, t_n$ are variables. A clause is* normal *if every compound term in C is normal. A clausal set N is* normal *if every clause in N is normal.*

Definition 11. *A compound term $f(t_1, \ldots, t_n)$ is* unique *if each pair of terms in $t_1, \ldots, t_n$ is a pair of distinct variables. A clause C is* unique *if every compound term in C is unique. A clausal set N is* unique *if every compound term in N is unique.*

In this thesis a new notion *strong compatibility* is introduced.

Definition 12. *A clause C is* strongly compatible *if all compound terms in C are compatible, and a clausal set N is* strongly compatible *is each clause in N is strong compatible.*

A strongly compatible clause is both linear and compatible. By generalising this claim to clausal sets, we have the following statement.

Lemma 3.2. *Let N be a strongly compatible clausal set. Then, N is locally compatible and locally linear.*

Proof. By **Definitions 8**, **9** and **12**. □

For a successful back-translation the pre-conditions are stated as follows.

Theorem 3.1 ([Eng96, Chapter 5])**.** *Let N be a first-order clausal set. Then, N can be unskolemised into a first-order formula (with equality) if N is normal, unique, globally linear and globally compatible.*

Back-translation rules

Rules that help the back-translation steps are the *variable renaming rule* **Rename**, the *term abstraction rule* **Abstract** and the *unskolemisation rule* **Unsko**.

A term t is abstracted from a clause C using

The Abstract rule

$$\frac{N \cup \{C(t)\}}{N \cup \{C(y) \vee t \not\approx y\}}$$

if y does not occur in $C(t)$.

A variable x of a clause C is rename to a distinct variable using

The Rename rule

$$\frac{N \cup \{C(x)\}}{N \cup \{C(y)\}}$$

if each occurrences of x in $C(x)$ is replaced by y, and y does not occur in $C(x)$.

A clausal set N is back-translated into a first-order formula using

The Unsko rule

$$\frac{N}{\exists \overline{x}_1 \forall \overline{x}_2 \exists \overline{x}_3 \forall \overline{x}_4 F}$$

if the following conditions are satisfied.

1. N is a normal, unique, globally linear and globally compatible clausal set.
2. $\overline{x}_1$ and $\overline{x}_3$ represent the restored Skolem constants and Skolem functions, respectively.
3. $\overline{x}_1$ and $\overline{x}_3$ do not occur in N, and $\overline{x}_2$ and $\overline{x}_4$ are variables in N.
4. F is a first-order formula without Skolem symbols.

The challenge of applying the **Unsko** rule to a clausal set N is not simply about computing a correction conclusion, but it is more about ensuring that N satisfies 1. in the **Unsko** rule, so that N can be unskolemised into a first-order formula. Given a clausal set N that is obtained by transforming a set of formulas to a clausal set N' and then saturating N', the **Unsko** rule restores first-order quantifications for N by eliminating Skolem symbols introduced during the Skolemisation step. We refer readers to [Eng96, Chapter 5] and [GSS08b] for more details of unskolemisation techniques.

Lemma 3.3 ([GSS08b]). *The **Abstract**, **Rename** and **Unsko** rules preserve logical equivalence.*

3.4 Saturation-based theorem proving

Substitution and unification

A *substitution* of terms for variables is a set $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where each x_i is a distinct variable and each t_i is a term, which is not identical to the corresponding variable x_i. We use lower-case Greek letters σ, θ and η to denote substitutions. By $E\sigma$, we denote the result of the *application of a substitution σ to an expression E*. $E\sigma$ is said to be an *instance* of E.

A *variable renaming* is a substitution σ such that $\sigma = \{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$ where $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ are variables. An expression E_1 is a *variant* of an expression E if there exists a variable renaming σ such that $E_1 = E\sigma$. We consider two clauses C_1 and C_2 be identical if C_1 is a variant of C_2. A substitution is called *grounding* if it substitutes all variables of an expression with ground terms. Given substitutions σ and θ, the *composition* $\sigma\theta$ denotes that for each variable x, $x\sigma\theta = (x\sigma)\theta$.

A substitution σ is a *unifier* of a set $\{E_1, \ldots, E_n\}$ of expressions if and only if $E_1\sigma = \ldots = E_n\sigma$. The set $\{E_1, \ldots, E_n\}$ is said to be *unifiable* if there is a unifier for it. A unifier σ of a set $\{E_1, \ldots, E_n\}$ of expressions is a *most general unifier (mgu)* if and only if for each unifier θ for the set, there exists a substitution η such that $\theta = \sigma\eta$. A unifier σ is a *simultaneous mgu* of two sequence $E_1, \ldots, E_n$ and $E'_1, \ldots, E'_n$ of expressions (where $n > 1$), if σ is an mgu for each pair E_i and E'_i. By $\sigma = \mathrm{mgu}(E \doteq E')$, we denote that σ is an mgu of expressions E and E'. By $\sigma = \mathrm{mgu}(E_1 \doteq E'_1, \ldots, E_n \doteq E'_n)$ (where $n > 1$), we denote that σ is a simultaneous mgu of two sequences $E_1, \ldots, E_n$ and $E'_1, \ldots, E'_n$ of expressions.

Orderings

Let S be a set. A binary relation R on S is a subset of $S \times S$. A *partial ordering* $\geq$ on a set S is a reflexive, antisymmetric and transitive binary relation. A *strict partial ordering* $\succ$ on a set S is an asymmetric and transitive binary relation. A strict ordering $\succ$ is *total* on a set S if for any two distinct elements x and y in S, either $x \succ y$ or $y \succ z$. A strict ordering $\succ$ is *well-founded* on a set S if there is no infinite chain $x_1 \succ x_2 \succ \ldots$ of elements in S.

We use $M(x)$ to denote the number of occurrences of variable x in a multiset M. A strict partial ordering $\succ$ on a set S can be extended to a *multiset ordering* $\succ^m$ on (finite) multisets over S as follows. Let M_1 and M_2 be two multisets. Then $M_1 \succ^m M_2$ if i) $M_1 \neq M_2$, and ii) if $M_2(x) > M_1(x)$ then $M_1(y) > M_2(y)$ for some $y \succ x$.

A binary relation $\rightarrowtail$ on expressions is *stable under contexts* if $E_1 \rightarrowtail E_2$ implies $E[E_1] \rightarrowtail E[E_2]$ for all expressions E, E_1 and E_2. A binary relation $\rightarrowtail$ is *stable under substitutions* (*liftable*) if $E_1 \rightarrowtail E_2$ implies $E_1\sigma \rightarrowtail E_2\sigma$ for all expressions E_1 and E_2, and any substitution σ. A binary relation $\rightarrowtail$ is a *rewrite relation* if $\rightarrowtail$ is stable under contexts and stable under substitutions.

An ordering $\succ$ has the *subterm property* if $E[E_1] \succ E_1$, for all for all expressions E and proper subexpressions E_1 of E. A *subterm ordering* is an ordering of a rewrite relation. An ordering $\succ$ is a *reduction ordering* if $\succ$ is a well-founded rewrite ordering. An ordering $\succ$ is a *simplification ordering* if $\succ$ is a reduction ordering with the subterm property.

An ordering $\succ$ on literals is *admissible* if

1. $\succ$ is well-founded and total on ground literals,
2. $\succ$ is stable under substitutions,
3. $\neg A \succ A$ for all ground atoms A,
4. if $B \succ A$, then $B \succ \neg A$ for all ground atoms A and B.

An ordering $\succ$ on literals can be extended to clauses by extending $\succ$ to clauses.

Let $\succ$ be an ordering, called a *precedence*, on the given set of function symbols, predicate symbols and logical symbols. Then based on this precedence, a *lexicographic path ordering* $\succ_{lpo}$ is defined as follows: $s \succ_{lpo} t$ if and only if

1. $t \in \text{var}(s)$ and $s \neq t$, or
2. $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and
   (a) $s_i \succeq_{lpo} t$ for some i with $1 \leq i \leq m$, or
   (b) $f \succ_{lpo} g$ and $s \succ_{lpo} t_j$ for all j with $1 \leq j \leq n$, or
   (c) i) $f = g$, and ii) for some j, we have $(s_1, \ldots, s_{j-1}) = (t_1, \ldots, t_{j-1})$, $s_j \succ_{lpo} t_j$, and iii) $s \succ_{lpo} t_k$, for all k with $j < k \leq n$.

If the precedence $\succ$ of a lexicographic path ordering is well-founded, then $\succ$ is a simplification ordering. A lexicographic path ordering $\succ_{lpo}$ over a total precedence is admissible if predicate symbols have higher precedence than logical connectives, which have higher precedence than $\top$ and $\bot$.

The ordered resolution calculus

In this section, we give fundamentals of a *saturation-based inference system*, based on the ordered resolution framework of [BG01, BG97]. The resolution calculus in the framework of [BG01, BG97] employs *admissible orderings* and *selection functions* as its refinement.

Let $\succ$ be an admissible ordering. Then we define *maximality* of a literal in a clause as follows.

- A ground literal L is called *maximal with respect to a ground clause* C if and only if for all L' in C, $L \succeq L'$.
- A ground literal L is called *strictly maximal with respect to a ground clause* C if and only if for all L' in C, $L \succ L'$.

- A non-ground literal L is *(strictly) maximal with respect to a clause* C if and only if there is some ground substitution σ such that $L\sigma$ is (strictly) maximal respect to $C\sigma$, that is for all L' in C, $L \geq L'$ ($L > L'$).

Let C be a clause. Then the *selection function* Select(C) is a mapping of a multiset of negative literals in C, and literals returned by Select(C) are the *selected literals*. There is no restriction imposed on selection functions. An *eligible literal* is either a (strictly) maximal literal or a selected literal. In this thesis, we annotate the (strictly) maximal literal L with 'stars' as in L^* and 'box' the selected literal L as in $\boxed{L}$.

We use the notation **Satu** to denote a resolution-based inference system that is parameterised by admissible orderings and selection functions. The **Satu** system consists of the *deduction rule* **Deduce**, the *positive factoring rule* **Fact**, the *selection-based ordered resolution rule* **Res** and the *deletion rule* **Delete**. In the **Fact** and **Res** rules, the conclusion are called a *factor* and a *resolvent* of its premises, respectively.

A saturation is deduced by

The Deduce rule (for clauses without equality)

$$\frac{N}{N \cup \{C\}}$$

if C is a conclusion of either the **Fact** or **Res** rule of clauses in N.

Factors are derived using

The Fact rule

$$\frac{C \vee A_1^* \vee A_2}{(C \vee A_1)\sigma}$$

if the following conditions are satisfied.

1. Nothing is selected in $C \vee A_1 \vee A_2$.
2. $A_1\sigma$ is $\succ$-maximal with respect to $C\sigma$.
3. $\sigma = \mathrm{mgu}(A_1 \doteq A_2)$

Resolvents are computed using

The Res rule

$$\frac{B_1^* \vee D_1,\ \ldots,\ B_n^* \vee D_n \qquad \boxed{\neg A_1 \vee \ldots \vee \neg A_n} \vee D}{(D_1 \vee \ldots \vee D_n \vee D)\sigma}$$

if the following conditions are satisfied.

1. No literal is selected in $D_1, \ldots, D_n$, and $B_1\sigma, \ldots, B_n\sigma$ are strictly $\succ$-maximal with respect to $D_1\sigma, \ldots, D_n\sigma$, respectively.

2a. If $n = 1$, then i) either $\neg A_1$ is selected, or nothing is selected in $\neg A_1 \vee D$ and $\neg A_1\sigma$ is $\succ$-maximal with respect to $D\sigma$, and ii) $\sigma = \mathrm{mgu}(A_1 \doteq B_1)$, or

2b. if $n > 1$, then $\neg A_1, \ldots, \neg A_n$ are selected and $\sigma = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$.

3. All premises are variable disjoint.

For decidability, we minimally need the following deletion rule.

The Delete rule

$$\frac{N \cup \{C\}}{N}$$

if C is a tautology, or N contains a variant of C.

In the **Res** rule, the premises $B_1 \vee D_1, \ldots, B_n \vee D_n$ are called the *positive premises* (*side premises*), and the premise $\neg A_1 \vee \ldots \vee \neg A_n \vee D$ is called the *negative premise* (*main premise*). If there is only one positive premise and one negative premise in the **Res** rule, we say it is a *binary resolution rule*.

The ordering refinement in inference rules can be applied by either *a prior checking* or *a posterior checking*. Let C be a premise in an inference rule, σ be the mgu in the rule and $\succ$ be the ordering refinement. Then if the maximal literal is determined in $C\sigma$, we say that $\succ$ is applied by *a posteriori checking*. If the maximal literal is determined in C, then $\succ$ is applied using *a prior checking*. In the **Fact** and **Res** rules, orderings are applied by a posterior checking.

The performance of a resolution-based inference system replies on sophisticated yet powerful *standard redundancy elimination* techniques. The **Satu** system only employs the **Deduce** rule, as it is sufficient for the results of this thesis.

Let N be a ground clausal set. A ground clause C is *redundant with respect* *to* N if there exists $C_1, \ldots, C_n$ in N such that $C_1, \ldots, C_n \models C$ and $C \succ C_i$ for each i with $1 \le i \le n$. Let N be a clausal set. Then a ground clause C is *redundant with respect to* N if there exists ground instances $C_1\sigma, \ldots, C_n\sigma$ of clauses $C_1, \ldots, C_n$ in N such that $C_1\sigma, \ldots, C_n\sigma \models C$ and $C \succ C_i\sigma$ for each i with $1 \le i \le n$. A non-ground clause C is *redundant with respect to* N if every ground instance of C is redundant with respect to N. Let C be a distinguished premise, $C_1, \ldots, C_n$ be other premises and D a conclusion in an inference **I**. Then *the **I** inference is redundant with respect to* N if there exist clauses $D_1, \ldots, D_k$ in N that are smaller than C such that $C_1, \ldots, C_n, D_1, \ldots, D_k \models D$. A clausal set N is *saturated up to redundancy with respect to an inference system* **R** if all inferences in the **R** inference system with non-redundant premises are redundant with respect to N.

A *derivation relation* $\triangleright$ is a binary relation defined on sets of clauses. Let N_1 and N_2 be two clausal set. By $N_1 \triangleright N_2$ on an inference system **R**, we mean that by using the rules in the **R** system to add conclusions or eliminate redundancy in clauses of N_1, we obtain N_2. A *theorem proving derivation* (*derivation*) on an inference system **R** is a sequence $N_1 \triangleright N_2 \triangleright \ldots$ of derivation.

The refutational completeness of the **Satu** system is given as follows.

Theorem 3.2 ([BG01, Theorem 5.5]). *If a set N of first-order clauses is saturated up to standard redundancy under the **Satu** system, then N is unsatisfiable if and only if it contains a contradiction.*

The soundness of the **Satu** system is obvious as it consists of sound rules.

Theorem 3.3. *The **Satu** system is a sound system for general first-order clausal logic.*

For the decidability results of this thesis the *separation rule* **Sep** rule is used. A clause can be separated by

The Sep rule

$$\frac{N \cup \{C \vee D\}}{N \cup \{C \vee P(\overline{x}), \neg P(\overline{x}) \vee D\}}$$

if the following conditions are satisfied.

1. C and D are non-empty subclauses.
2. $\mathrm{var}(C) \not\subseteq \mathrm{var}(D)$ and $\mathrm{var}(D) \not\subseteq \mathrm{var}(C)$.
3. $\mathrm{var}(C) \cap \mathrm{var}(D) = \overline{x}$.
4. Predicate symbol P does not occur in $N \cup \{C \vee D\}$.

The **Sep** rule is introduced in [SH00] to decide satisfiability of fluted logic. This rule is also referred to as 'splitting through new predicate symbol' in [Kaz06, Section 3.5.6].

The *split rule* **Split** is very similar to the **Sep** rule. A derivation sequence is branched to a derivation tree by

The Split rule

$$\frac{N \cup \{C \vee D\}}{N \cup \{C\} \mid N \cup \{D\}}$$

if the following conditions are satisfied.

1. C and D are non-empty subclauses.
2. C and D are variable-disjoint.

In the above **Split** rule, the symbol '|' in the **Split** conclusions means that the sequence of derivation on $N \cup \{C \vee D\}$ is split into two branches $N \cup \{C\}$ and $N \cup \{D\}$.

One can regard the **Sep** rule as a generalisation of the **Split** rule. Suppose that in the **Sep** premise $N \cup \{C \vee D\}$, the subclauses C and D are variable disjoint. Then using a fresh predicate symbol p, the **Sep** rule derives $N \cup \{C \vee p, \neg p \vee D\}$ from $N \cup \{C \vee D\}$. This implies that the **Sep** rule can be regarded as a generalisation of the **Split** rule by using a new predicate symbol [RV01a]. Compared to the **Sep** rule, the **Split** rule splits $N \cup \{C \vee D\}$ to two branches $N \cup \{C\}$ and $N \cup \{D\}$. This requires backtracking when an empty clause is found in one branch. Hence, using the **Split** rule makes the saturation procedure non-deterministic. However, the **Split** rule has an advantage that one can use the subsumption elimination rule [BG01] to remove clauses in the forms of $C \vee C'$ and $D \vee D'$ in $N \cup \{C\}$ and $N \cup \{D\}$, respectively. The **Sep** conclusion $N \cup \{C \vee p, \neg p \vee D\}$ does not have this advantage because of the occurrences of the propositional symbol p.

The **Sep** rule is a sound rule. This is formally stated as:

Lemma 3.4 ([SH00, Theorem 3]). *The **Sep** premises $N \cup \{C \vee D\}$ are satisfiable if and only if the **Sep** conclusions $N \cup \{C \vee P(\overline{x}), \neg P(\overline{x}) \vee D\}$ are satisfiable.*

Proof. $\Leftarrow$: By respectively making $P(\overline{x})$ and $\neg P(\overline{x})$ in $C \vee P(\overline{x})$ and $\neg P(\overline{x}) \vee D$ eligible, applying resolution to $C \vee P(\overline{x})$ and $\neg P(\overline{x}) \vee D$ derives $C \vee D$. Hence, for any interpretation I such that $I \models N \cup \{C \vee P(\overline{x}), \neg P(\overline{x}) \vee D\}$, it is the case that $I \models N \cup \{C \vee D\}$.

$\Rightarrow$: Suppose I is a model of $N \cup \{C \vee D\}$. We aim to prove that an extension I' of I satisfies that $I' \models N \cup \{C \vee P(\overline{x}), \neg P(\overline{x}) \vee D\}$. As I' is an extension of I, $I' \models N \cup \{C \vee D\}$. We next prove that $I' \models C \vee P(\overline{x})$ and $I' \models \neg P(\overline{x}) \vee D$.

Suppose $\overline{x}$ is a sequence of variables $x_1, \ldots, x_n$, $\overline{s}$ is a sequence of ground terms $s_1, \ldots, s_n$. Further suppose σ is a ground substitution that substitutes $x_1, \ldots, x_n$ through $\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$. Let Σ be a set of all possible ground substitutions of σ. Then we interpret $P(s_1, \ldots, s_n)$ as follows. An interpretation I' is a model of $P(s_1, \ldots, s_n)$ if and only if $I \models D\sigma$ for all σ in Σ.

Let θ be an arbitrary ground substitution. We aim to prove that

$$I' \models C\theta \vee P(\overline{x}\theta), \tag{3.1}$$

$$I' \models \neg P(\overline{x}\theta) \vee D\theta. \tag{3.2}$$

We distinguish two cases:

i: Assume $I' \models P(\overline{x}\theta)$. By the interpretation of $P(\overline{x}\theta)$, $I' \models D\theta$, hence (3.1)–(3.2) hold.

ii: Suppose $I' \not\models P(\overline{x}\theta)$. Immediately (3.2) holds. We prove (3.1) by contradiction. Suppose there exists a ground substitution θ' such that i) θ' and θ coincide on substituting $\overline{x}$ and ii) $I' \not\models C\theta'$. By our interpretation of $P(\overline{x}\theta)$, there exists a ground substitution θ'' such that i) θ'' and θ coincide on substituting $\overline{x}$ and ii) $I' \not\models D\theta''$. Since $I' \not\models C\theta'$ and $I' \not\models D\theta''$ and θ' and θ'' coincide on substituting common variables $\overline{x}$ of C and D, $I' \not\models C\theta'\theta'' \vee D\theta'\theta''$. This contradicts that $I' \models N \cup \{C \vee D\}$.

W.l.o.g. the proof can be generalised to the cases when C or D is negative. □

The ordered superposition calculus

Now we introduce superposition calculus to reason equality literals. As for the purpose of this thesis, a weaker form of superposition calculus, namely the paramodulation calculus, is sufficient. The paramodulation calculus are also in the framework of [BG98].

For the ordering purpose for equality, non-equational literals $P(t_1, \ldots, t_n)$ with P a non-equational predicate symbol, are treated as $P(t_1, \ldots, t_n) \approx \mathbf{tt}$ with

tt a distinguished constant. In any admissible ordering $\succ$, **tt** is always the minimal constant. Admissible orderings $\succ$ are extended to multiset orderings $\succ^m$ by comparing literals in a way such that equality literals $s \approx t$ are regarded as $\{s, t\}$ and inequality literals $s \not\approx t$ are regarded as $\{s, t, \mathbf{tt}\}$, respectively.

We use the notation **Satu**$_\approx$ to denote the **Satu** system with the *equality factoring rule* **E-Fact**, the *equality resolution rule* **E-Res** and the *ordered paramodulation rule* **Para** and a revised **Deduce** rule. We assume that equality literals are oriented: whenever we write $s \approx t$ and $s \not\approx t$, it is the case that $s \succeq t$.

A derivation is computed using

The Deduce rule (for clauses with equality)

$$\frac{N}{N \cup \{C\}}$$

if C is a conclusion of either the **Fact**, or **Res**, or **E-Fact**, or **E-Res** or the **Para** rule of clauses in N.

Conclusions of the ordered paramodulation rule is computed using

The Para rule

$$\frac{t_1 \approx u \vee D_1 \qquad L[t_2] \vee D_2}{(L[u] \vee D_1 \vee D_2)\sigma}$$

if the following conditions are satisfied.

1. Nothing is selected in $D_1\sigma$ and $(t_1 \approx u)\sigma$ is strictly $\succ^m$-maximal with respect to $D_1\sigma$.
2. If $L[t_2]$ is positive, $L[t_2]\sigma$ is strictly $\succ^m$-maximal with respect to $D_2\sigma$, or else $L[t_2]\sigma$ is either selected or $\succ^m$-maximal with respect to $D_2\sigma$.
3. t_2 is not a variable.
4. $u\sigma \not\succeq t_1\sigma$.
5. $\sigma = \mathrm{mgu}(t_1 \doteq t_2)$.
6. Premises are variable disjoint.

In the **Para** rule, the premises $t_1 \approx u \vee D_1$ and $L[t_2] \vee D_2$ are called the *left premise* and the *right premise*, respectively.

Conclusions of the equality factoring rule is computed using

The E-Fact rule

$$\frac{t_1 \approx u \vee t_2 \approx v \vee D}{(u \not\approx v \vee t_1 \approx v \vee D)\sigma}$$

if the following conditions are satisfied.

1. Nothing is selected in D and $(t_1 \approx u)\sigma$ is $\succ^m$-maximal with respect to $(t_2 \approx v \vee D)\sigma$.
2. $u\sigma \not\succeq t_1\sigma$.
3. $\sigma = \mathrm{mgu}(t_1 \doteq t_2)$.

Conclusions of the equality resolution rule is computed using

The E-Res rule

$$\frac{t_1 \not\approx t_2 \vee D}{D\sigma}$$

if the following conditions are satisfied.

1. Either $(t_1 \not\approx t_2)\sigma$ is selected or it is $\succ^m$-maximal with respect to $D\sigma$.
2. $\sigma = \mathrm{mgu}(t_1 \doteq t_2)$.

Theorem 3.4. *The **Satu**$_\approx$ system is sound and refutationally complete for general first-order clausal logic with equality.*

Proof. It can easily be checked that the **E-Fact**, **E-Res** and **Para** rules preserve satisfiability, as they are standard rules in [BG90]. By [BG90, Theorem 1], the **Satu**$_\approx$ system is refutationally complete for first-order clausal logic with equality. □

Chapter 4

The decision procedure for answering BCQs in GF

In this chapter, we tackle the problem of answering BCQs for guarded formulas. This is formally stated as:

Problem 4. *Given a set Σ of formulas in GF and a union q of BCQs, can a saturation-based procedure decide whether $\Sigma \models q$?*

This chapter is constructed as follows. **Section** 4.1 describes the clausification process that transforms guarded formulas and BCQ into a suitable clausal form, namely guarded clauses and query clauses, respectively. **Section** 4.2 then gives a **P-Res** resolution inference system **Inf**. Based on the **Inf** system, **Section** 4.3 then devises the top-variable inference system **T-Inf**$^{\mathsf{GQ}}$, particularly for the guarded clauses and query clauses. **Section** 4.4 then formally proves that the **T-Inf**$^{\mathsf{GQ}}$ system decides satisfiability of the guarded clauses, and **Section** 4.5 presents the procedure of handling the query clauses. Finally combining the results of **Sections** 4.1–4.5, **Section** 4.6 gives a saturation-based decision procedure for answering BCQs for GF.

4.1 Clausifying GF and BCQs

In this section, we aim to reduce the BCQ answering problem for GF to a satisfiability checking problem for a specific clausal class, and we use a customised form of *clausal normal form transformation* to achieve this goal.

We use the notation $\textbf{Trans}^{\textsf{GF}}$ to denote our *clausal normal form transformation* for guarded formulas and BCQs. In the first step, a union of BCQs is simply negated to obtain *query clauses*. The second step transforms guarded formulas to a set of *guarded clauses*. Recall the definition of GF from **Section 2.1**.

Definition 1. *The* guarded fragment (GF) *is a fragment of FOL without function symbols, 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 connectives.*
4. *Let F be a guarded formula and G an atom. Then $\exists \overline{x}(G \wedge F)$ and $\forall \overline{x}(G \rightarrow F)$ belong to GF if all free variables of F occur in G.*

Note that we assume that all free variables in guarded formulas are existentially quantified as we are focusing on checking satisfiability.

Using sample guarded formulas

$$F = [\exists x(A(x,y) \wedge \forall z(B(x,z) \rightarrow \exists u R(z,u)))],$$

the second step of the $\textbf{Trans}^{\textsf{GF}}$ process is detailed next.

1. Add existential quantifiers to all free variables of F, and by the **NNF** rules, transforming F to negation normal form, obtaining

$$F_1 = \left[\begin{array}{ll} \exists yx(& A(x,y) \wedge \forall z(\\ & \neg B(x,z) \vee \exists u R(z,u)) \quad) \end{array} \right].$$

2. By introducing predicate symbols P (and respective literals $P(\cdots)$), applying the **Trans** rules for each universally quantified subformula of F_1. Then we obtain

$$F_2 = \left[\begin{array}{lll} \exists yx(& A(x,y) \wedge P(x) &)\wedge \\ \forall x(& \neg P(x) \vee \forall z(\neg B(x,z) \vee \exists u R(z,u)) &) \end{array} \right].$$

We say that

- $\exists yx(A(x,y) \wedge P(x))$ is the *replacing formula of* F_1, and
- $\forall x(\neg P(x) \vee \forall z(\neg B(x,z) \vee \exists u R(z,u)))$ is the *definition formula of* P.

3. Transform each immediate subformula of F_2 to prenex normal form, and then applying the **Skolem** rule to the resulting formula. By introducing Skolem constants a, b and a Skolem function $f(x,z)$, we obtain

$$F_3 = \begin{bmatrix} & A(a,b) & \wedge \\ & P(a) & \wedge \\ \forall xz(& \neg P(x) \vee \neg B(x,z) \vee R(z,f(x,z)) &) \end{bmatrix}.$$

4. Drop universal quantifiers of F_3, and then by the **CNF** rules, F_3 is transformed to a set of *guarded clauses*

$$\{A(a,b),\ P(a),\ \neg P(x) \vee \neg B(x,z) \vee R(z,f(x,z))\}$$

The *guarded*, *Horn guarded* and *query clauses* are formally defined as follows.

Definition 13. *A* guarded clause *C is a simple and covering clause satisfying the following conditions:*

1. *C is either a ground clause, or*
2. *C contains a negative flat literal* $\neg G$ *such that* $\mathrm{var}(C) = \mathrm{var}(G)$.

A Horn guarded clause (HG clause) *is a guarded clause containing at most one positive literal.*

We call the literal $\neg G$ in *2.* of **Definition 13** the *guard* of the guarded clause C. A clause is *guarded* if it contains a guard.

Definition 14. *A* query clause *is a flat and negative clause.*

In *2.* of **Definition 13**, the literal $\neg G$ is called the *guard* of the clause C. A query clause is not necessarily a guarded clause, and vice-versa. For example, $\neg A(x,y) \vee B(f(x,y))$ is guarded but not a query clause, and $\neg A_1(x,y) \vee \neg A_2(y,z)$ is a query clause, but not guarded. The class of guarded clauses is more expressive than GF, since compound terms are allowed in the clausal class, but not in GF.

As the **Trans**$^{\mathsf{GF}}$ process only provides essential steps, one can use more exhaustive structural transformations to transform guarded formulas to a simpler form of guarded clauses and obtain guarded clauses in a more efficiently way. For example, guarded formulas are transformed to guarded clauses with at most three literals in [Kaz06, Pages 103–104]. Moreover by applying the **Trans**$^{\mathsf{GF}}$ rules to conjunctive formulas that are disjunctively connected, one can avoid the exponential-time blow-up caused by distributing disjunctions to conjunctions. For example, it takes exponential steps for **Trans**$^{\mathsf{GF}}$ process to transform the guarded formula

$$F = \forall xy(G(x, y) \to (A(x) \land B(y)) \lor (A(y) \land B(x)))),$$

to the guarded clauses

$$\begin{array}{ll} \neg G(x, y) \lor A(x) \lor A(y), & \neg G(x, y) \lor B(y) \lor A(y), \\ \neg G(x, y) \lor A(x) \lor B(x), & \neg G(x, y) \lor B(y) \lor B(x). \end{array}$$

However in F, using new predicate symbols $P_1(x, y)$ and $P_2(x, y)$ for $A(x) \land B(y)$ and $A(y) \land B(x)$, respectively, the distribution of disjunctions to conjunctions can be avoided. Then F is transformed into the guarded clauses

$$\begin{array}{l} \neg G(x, y) \lor P_1(x, y) \lor P_2(x, y), \\ \neg P_1(x, y) \lor A(x) \lor B(y), \\ \neg P_2(x, y) \lor A(y) \lor B(x). \end{array}$$

Note that by i) renaming universal quantified subformulas, ii) transforming formulas to prenex normal form and then applying Skolemisation to the resulting formulas, the **Trans**$^{\mathsf{GF}}$ process intentionally introduces Skolem functions of a higher arity. To be specific i)–ii) ensure that a guarded clause C has the *covering* property, i.e., any compound term in C contains exactly the same set of variables as C. This property is essential to guarantee termination of our BCQ answering procedures for GF. Also i)–ii) ensure compound terms in the guarded clause C are *aligned* (see **Section 5.1**), i.e., all compound terms in C share the same sequence of variables (i.e. the *strong compatibility property*). This property makes our back-translation procedure possible.

Lemma 4.1. *Applying the* ***Trans***$^{\mathsf{GF}}$ *process to a guarded formula transforms it into a set of guarded clauses.*

Proof. Suppose F is a guarded formula. In the **Trans**$^{\mathsf{GF}}$ process, 1.–2. use new predicate symbols (and literals) to rename universally quantified formulas in F. W.l.o.g. suppose P is the newly introduced predicate symbol, F_1 is the definition formula of P, and F' is the replacing formula of F. Now we show that 3.–4. transform F_1 and F' into guarded clause. Because F' is an existentially quantified sentence, skolemising F' transforms it into (a set of) flat ground clauses (if conjunctions occur in F'), which are guarded clauses. F_1 can be represented as

$$\forall \overline{x}(P(\overline{x}) \rightarrow \forall \overline{y}(G(\overline{x}, \overline{y}) \rightarrow \phi(\overline{y})))$$

where $\phi(\overline{y})$ is a formula of literals and existentially quantified guarded formulas that are connected by Boolean connectives. Note that $\phi(\overline{y})$ contains no universal quantifications. By *4.* in the **Trans**$^{\mathsf{GF}}$ process, F_1 is simplified as

$$F_1' = \forall \overline{x}\overline{y}(\neg P(\overline{x}) \vee \neg G(\overline{x}, \overline{y}) \vee \phi(\overline{y})).$$

Suppose C is a clause obtained from F_1'. 1) The literal $\neg G(\overline{x}, \overline{y})$ is a guard of C as $\text{var}(G) = \text{var}(F)$. 2) For any existential quantified variable z in $\phi(\overline{y})$, z is Skolemised into a flat compound term only containing $\overline{x}$ and $\overline{y}$. 3) Since F_1' is free of function symbols, C contains no nested compound terms. By 1)–3), C is simple, covering and contains the guard $\neg G$, thus C is a guarded clause. □

We use GQ to denote the class of guarded clauses and query clauses.

Theorem 4.1. *The **Trans**$^{\mathsf{GF}}$ process reduces the problem of BCQ answering for GF to that of deciding satisfiability of the GQ clausal class.*

Proof. Suppose $q = q_1 \vee \ldots \vee q_n$ is a union of BCQs, Σ is a set of guarded formulas, and D is a set of ground atoms. Since ground atoms D are in GF, the problem of checking whether $\Sigma \cup \mathrm{D} \models q$ is reduced to that of $\Sigma \models q$. This problem is the same as the problem of checking unsatisfiability of $\Sigma \cup \{\neg q_1, \ldots, \neg q_n\}$. By the definition of the union of BCQs, $\{\neg q_1, \ldots, \neg q_n\}$ is a set of query clauses. By **Lemma 4.1**, Σ is transformed to a set of guarded clauses. □

4.2 The resolution-based P-Res inference system

In this section, we presents the first **P-Res** inference system **Inf**, which provides a basis for the decision procedures in this thesis. The **Inf** system is built on the **Satu** system from **Section 3.4**, however unlike the **Satu** system, the **Inf** system generalises the **Res** rule to a novel *partial selection-based ordered resolution rule* **P-Res**. Therefore we call this system a ***P-Res** system*. The **P-Res** rule allows us to choose a desirable resolvent from a set of the potential *partial resolvents*. In this section, we extensively discuss the **P-Res** rule and formally prove the soundness and refutational completeness of the **Inf** system.

The **Inf** system contains the following rules: the *deduction rule* **Deduce**, the *positive factoring rule* **Fact**, the *partial selection-based resolution rule* **P-Res** and the *deletion rule* **Delete**.

A saturation is deduced using

The Deduce rule (for clauses without equality)

$$\frac{N}{N \cup \{C\}}$$

if C is a conclusion of the **P-Res** or **Fact** rule of clauses in N.

Factors are computed using

The Fact rule

$$\frac{C \vee A_1^* \vee A_2}{(C \vee A_1)\sigma}$$

if the following conditions are satisfied.

1. Nothing is selected in $C \vee A_1 \vee A_2$.
2. $A_1\sigma$ is $\succ$-maximal with respect to $C\sigma$.
3. $\sigma = \mathrm{mgu}(A_1 \doteq A_2)$

For decidability, we use the following deletion rule.

The Delete rule

$$\frac{N \cup \{C\}}{N}$$

if C is a tautology, or N contains a variant of C.

A partial selection-based resolution **P-Res** computes resolvents using

The P-Res rule

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

if the following conditions are satisfied.

1. No literal is selected in $D_1, \ldots, D_n$ and $B_1\sigma, \ldots, B_n\sigma$ are strictly $\succ$-maximal with respect to $D_1\sigma, \ldots, D_n\sigma$, respectively.

2a. If $n = 1$, i) either $\neg A_1$ is selected, or nothing is selected in $\neg A_1 \vee D$ and $\neg A_1\sigma$ is maximal with respect to $D\sigma$, and ii) $\sigma = \text{mgu}(A_1 \doteq B_1)$ or

2b. if $n > 1$ and there exists an mgu σ' such that $\sigma' = \text{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$, then $\sigma = \text{mgu}(A_1 \doteq B_1, \ldots, A_m \doteq B_m)$ where $m \leq n$.

3. All premises are variable disjoint.

Only essential rules are presented in the **Inf** system. The **Inf** system is devised in line with the resolution framework of [BG01], therefore more sophisticated *simplification rule* (such as the *condensation rule*) and redundant elimination techniques (such as *forward* and *backward subsumption elimination*) [BG01, Section 4.3], can be immediately added to the **Inf** system.

In the **P-Res** rule, the distinguished premise

$$\neg A_1 \vee \ldots \vee \neg A_m \vee \ldots \vee \neg A_n \vee D$$

is called the *main premise* (*negative premise*) and the other premises

$$B_1 \vee D_1, \ldots, B_m \vee D_m, \ldots, B_n \vee D_n$$

are called the *side premises* (*positive premises*). The **P-Res** rule generalises the *hyper-resolution rule* in [BG01, Section 6.2], since in the **P-Res** rule, side premises and the subclause D in the main premise are not necessarily positive. This relaxed condition implicitly ensures that the **P-Res** rule is a natural generalisation of the *binary resolution rule* (the *ordered resolution rule with selection*) in [BG01], if there exists exactly one eligible literal in the main premise. By the *binary* *resolution rule*, we mean a resolution rule with only one positive premise and one negative premise.

The **P-Res** rule is a form of 'partial' selection-based resolution rule. In the conditions of the **P-Res** rule, 2b. requires the existence of an mgu between $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$. This implies that one can perform a selection-based resolution inference on

$$B_1 \vee D_1,\ \ldots,\ B_n \vee D_n,\ \neg A_1 \vee \ldots \vee \neg A_m \vee \ldots \vee \neg A_n \vee D$$

with $\neg A_1, \ldots, \neg A_n$ selected. However, instead of performing this selection-based resolution inference, we perform a partial selection-based resolution inference on

$$B_1 \vee D_1,\ \ldots,\ B_m \vee D_m,\ \neg A_1 \vee \ldots \vee \neg A_m \vee \ldots \vee \neg A_n \vee D.$$

This 'partial' inference on C and a subset of $C_1, \ldots, C_n$ makes a selection-based resolution inference on C and $C_1, \ldots, C_n$ redundant. This claim is formally proved in **Lemmas** 4.2–4.3, starting with considering ground first-order clauses.

Lemma 4.2 ([BG01, Pages 53–54] and [BG97, Page 28])**.** *Let the following rule present the* ***Res*** *rule of the* ***Satu*** *system for ground clauses.*

Res *(for ground clauses):* $\dfrac{A_1 \vee D_1,\ \ldots,\ A_n \vee D_n \qquad \neg A_1 \vee \ldots \vee \neg A_n \vee D}{D_1 \vee \ldots \vee D_n \vee D}$

if the following conditions are satisfied.

1. *No literals are selected in* $D_1, \ldots, D_n$ *and* $A_1, \ldots, A_n$ *are strictly* $\succ$*-maximal with respect to* $D_1, \ldots, D_n$*, respectively.*

2a. *If* $n = 1$*, then either* $\neg A_1$ *is selected, or nothing is selected in* $\neg A_1 \vee D$ *and* $\neg A_1$ *is* $\succ$*-maximal with respect to* D*, or*

2b. *if* $n > 1$*, then* $\neg A_1, \ldots, \neg A_n$ *are selected.*

3. *All premises are variable disjoint.*

Let $\{1, \ldots, n\}$ *be partitioned into two subsets* $\{i_1, \ldots, i_k\}$ *and* $\{j_1, \ldots, i_h\}$*, and*

*N be a clausal set. Then an **Res** inference is redundant in N if the 'partial conclusion'*

$$\neg A_{j_1} \vee \ldots \vee \neg A_{j_h} \vee D_{i_1} \vee \ldots \vee D_{i_k} \vee D$$

is implied by $A_1 \vee D_1, \ldots, A_n \vee D_n$ and finitely many clauses Δ in N that are smaller than $\neg A_1 \vee \ldots \vee \neg A_n \vee D$.

Proof. W.l.o.g., let the main premise of an **Res** inference be of the form

$$\neg A_1 \vee \ldots \vee \neg A_m \vee \neg A_{m+1} \vee \ldots \vee \neg A_n \vee D$$

and the 'partial conclusion' be of the form

$$\neg A_{m+1} \vee \ldots \vee \neg A_n \vee D_1 \vee \ldots \vee D_m \vee D$$

where $m < n$. When $m = n$, the statement trivially holds.

By the definition of redundant inference, this claim requires to prove that

$$A_1 \vee D_1, \ldots, A_n \vee D_n, \Delta \models D_1 \vee \ldots \vee D_n \vee D.$$

Firstly, in (4.1)–(4.7), we aim to prove that

$$A_1, \ldots, A_n, A_1 \vee D_1, \ldots, A_n \vee D_n, \Delta \models D_1 \vee \ldots \vee D_n \vee D.$$

By the assumption on 'partial conclusion',

$$A_1 \vee D_1, \ldots, A_n \vee D_n, \Delta \models \tag{4.1}$$

$$\neg A_{m+1} \vee \ldots \vee \neg A_n \vee D_1 \vee \ldots \vee D_m \vee D. \tag{4.2}$$

Suppose I is an interpretation of (4.1) and suppose $I \models A_1, \ldots, A_n$. Then

$$I \models A_1, \ldots, A_n, A_1 \vee D_1, \ldots, A_n \vee D_n, \Delta. \tag{4.3}$$

Since $I \models A_{m+1}, \ldots, A_n$ and $I \models$ (4.2), we obtain that

$$I \models D_1 \vee \ldots \vee D_m \vee D. \tag{4.4}$$

This implies that

$$A_1, \ldots, A_n, A_1 \vee D_1, \ldots, A_n \vee D_n, \Delta \tag{4.5}$$
$$\models D_1 \vee \ldots \vee D_m \vee D. \tag{4.6}$$

Since $m < n$, clause in (4.6) is a subclause of $D_1 \vee \ldots \vee D_n \vee D$. Hence:

$$\begin{aligned} &A_1, \ldots, A_n, A_1 \vee D_1, \ldots, A_n \vee D_n, \Delta \\ &\models D_1 \vee \ldots \vee D_n \vee D. \end{aligned} \tag{4.7}$$

Next, in (4.9)–(4.12), we aim to prove that

$$A_1 \vee D_1, \ldots, A_n \vee D_n, \Delta \models D_1 \vee \ldots \vee D_n \vee D. \tag{4.8}$$

We prove (4.8) by contradiction. Let I be an arbitrary model satisfying that

$$I \models A_1 \vee D_1, \ldots, A_n \vee D_n, \Delta, \tag{4.9}$$
$$\text{but } I \not\models D_1 \vee \ldots \vee D_n \vee D. \tag{4.10}$$

(4.10) implies $I \not\models D_1, \ldots, I \not\models D_n$, therefore, considering (4.9) we get that

$$I \models A_1, \ldots, A_n, \Delta. \tag{4.11}$$

By (4.9) and (4.11), we obtain

$$I \models A_1, \ldots, A_n, A_1 \vee D_1, \ldots, A_n \vee D_n, \Delta. \tag{4.12}$$

By (4.7), (4.12) implies $I \models D_1 \vee \ldots \vee D_n \vee D$, which refutes (4.10). □

Using **Lemma** 4.2, we prove that an **P-Res** inference makes its respective **Res** inference redundant, formally stated:

Lemma 4.3. *Let N a clausal set. Suppose the **Res** rule (for ground clauses) is applicable in N to the premises $C_1 = A_1 \vee D_1, \ldots, C_n = A_n \vee D_n$ and $C = \neg A_1 \vee \ldots \vee \neg A_n \vee D$. Suppose the 'partial conclusion' $R = D_{i_1} \vee \ldots \vee D_{i_k} \vee D'$ is obtained by performing the following inference on the main premise C and a subset of the side premises $C_1, \ldots, C_n$.*

$$\textbf{P-Res}: \quad \frac{A_{i_1} \vee D_{i_1}, \ \ldots, \ A_{i_k} \vee D_{i_k} \qquad \neg A_{i_1} \vee \ldots \vee \neg A_{i_k} \vee D'}{D_{i_1} \vee \ldots \vee D_{i_k} \vee D'}$$

if the following conditions are satisfied.

1. $\{A_{i_1} \vee D_{i_1}, \ldots, A_{i_k} \vee D_{i_k}\}$ *is a subset of* $\{C_1, \ldots, C_n\}$.
2. $\neg A_{i_1} \vee \ldots \vee \neg A_{i_k} \vee D'$ *is the same as the main premise* C.

Then the application of the **Res** *rule (for ground clauses) to* $C_1, \ldots, C_n$ *and* C *is redundant with respect to* $N \cup R$.

Proof. By maximality refinement, $A_{i_j} \succ D_{i_j}$ for all j such that $i \leq j \leq k$. Hence, R is smaller than $\neg A_{i_1} \vee \ldots \vee \neg A_{i_k} \vee D'$, thus R is smaller than C. By **Lemma** 4.2 and the fact that $C_1, \ldots, C_n, R \models R$, the specified application of the **Res** rule (for ground clauses) is redundant in $N \cup R$. □

In **Lemmas** 4.2–4.3, the **Res** and **P-Res** rules use admissible orderings and selection function as resolution refinements, therefore by the Lifting Lemma [BG01, Lemma 4.12], the result of **Lemma** 4.3 can be immediately lifted to general first-order clauses.

Suppose the **Res** rule is applicable to $C_1 = A_1 \vee D_1, \ldots, C_n = A_n \vee D_n$ and $C = \neg A_1 \vee \ldots \vee \neg A_n \vee D$. Then one derives a 'partial conclusion' by applying the **P-Res** rule to a subset of $\{C_1, \ldots, C_n\}$ and C, where a subset of $\{\neg A_1, \ldots, \neg A_n\}$ are resolved. We say this subset of $\{\neg A_1, \ldots, \neg A_n\}$ are the ***P-Res*** *eligible literal* (with respect to a **Res** inference to $C_1, \ldots, C_n$ and C). In this paper, we consider applications of the **P-Res** rule by focusing on finding appropriate **P-Res** eligible literals.

Now we give the main result of this section.

Theorem 4.2. *The* ***Inf*** *system is sound and refutationally complete for general first-order clausal logic.*

Proof. Compared to the **Satu** resolution system in **Section** 3.4, in the **Inf** system, the novel rule is the **P-Res** rule. The **P-Res** rule is sound as it is a resolution rule. Hence, the **Inf** system is sound. We know that the **Res** rule is a standard rule in the **Satu** system. By **Lemma** 4.3, a **P-Res** inference can be regarded as a form of redundancy elimination for its respective **Res** inference. Then the **Inf** system is refutationally complete for first-order clauses as the **Satu** system is refutationally complete for first-order clauses. □

4.3 The top-variable refinement

In this section, we give the top-variable refinement **T-Ref**$^{\mathsf{GQ}}$, so that the **Inf** system, equipped with the **T-Ref**$^{\mathsf{GQ}}$ refinement, decides satisfiability of the GQ clausal class. We use the notation **T-Inf**$^{\mathsf{GQ}}$ to denote the **Inf** system endowed with the **T-Ref**$^{\mathsf{GQ}}$ refinement.

As admissible orderings we use any lexicographic path ordering $\succ_{lpo}$ with a precedence in which function symbols are larger than constant, which are larger than predicate symbols. This requirement holds however for any admissible ordering (e.g., *Knuth-Bendix ordering* [KB83]) with the same precedence restriction.

Algorithm 1: Determining the (**P-Res**) eligible literals for GQ clauses

Input: A GQ clausal set N and a clause C in N
Output: The eligible literals or the **P-Res** eligible literals (with respect to a **Res** inference) in C

if C *is a ground clause* **then**
  return Max(C)
else if C *has negatively occurring compound-term literals* **then**
  return SelectNC(C)
else if C *has positively occurring compound-term literals* **then**
  return Max(C)
else if C *is a flat guarded clause* **then**
  return SelectG(C)
else **return** PResT(N, C)

Algorithm 1 specifies conditions for applying the **T-Ref**$^{\mathsf{GQ}}$ refinement to GQ clauses. The **T-Ref**$^{\mathsf{GQ}}$ refinement consists of the following functions.

- Max(C) returns the (strictly) $\succ_{lpo}$-maximal literal with respect to clause C.

- SelectNC(C) selects one of the negative compound-term literals in clause C.
- SelectG(C) selects one of the guards in clause C.
- PResT(N, C)
    1. either returns (selects) all negative literals of clause C, in the case that the **Res** rule is not applicable to C, in which all negative literals are selected (as the main premise), and clauses in N (as the side premises), or
    2. returns the *top-variable literals* (with respect to a **Res** inference) of clause C, in the case that the **Res** rule is applicable to C, in which all negative literals are selected (as the main premise), and clauses in N (as the side premises).

Algorithm 2 details the PResT function. By **Algorithm 1**, the PResT(N, C) takes a GQ clausal set N and a query clause C as inputs. Lines 2–4 aim to check whether the **Res** rule is applicable to $C_1, \ldots, C_n$ (occurring in N) and C with all negative literals selected. If the **Res** rule is applicable to $C_1, \ldots, C_n$ and C, then Line 5 uses the CompT($C_1, \ldots, C_n, C$) function to compute the **P-Res** eligible literals in C (with respect to an **Res** inference to $C_1, \ldots, C_n$ and C). In particular these **P-Res** eligible literals are called the *top-variable literals*, since they are computed by the so-called the *top-variable technique*. However if the **Res** rule is not applicable to $C_1, \ldots, C_n$ and C, all selected negative literals of C are returned, as shown in Line 6.

Algorithm 2: The PResT function

Input: A clausal set N and a clause C in N

Output: The eligible literals or the **P-Res** eligible literals (with respect to a **Res** inference) in C

1 **Function** PResT(N, C):
2   Select all negative literals in C
3   Find the side premises of C occurring in N, namely $C_1, \ldots, C_n$
4   **if** $C_1, \ldots, C_n$ *exist* **then**
5   **return** CompT($C_1, \ldots, C_n, C$)
6   **else** **return** all negative literals in C

Now we formally introduce the top-variable technique, given by the CompT function. Suppose in a **Res** inference, $C_1 = B_1 \vee D_1, \ldots, C_n = B_n \vee D_n$ are the side premises and $C = \neg A_1 \vee \ldots \vee \neg A_n \vee D$ is the main premise, in which $\neg A_1 \vee \ldots \vee \neg A_n$ are selected. Then the $\mathrm{CompT}(C_1, \ldots, C_n, C)$ function computes the *top variables* and the *top-variable literals* of C as follows.

1. Without producing or adding the resolvent, compute an mgu σ' for $C_1, \ldots, C_n$ and C such that $\sigma' = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$.

2. Compute the *variable ordering* $>_v$ and $=_v$ over the variables of $\neg A_1 \vee \ldots \vee \neg A_n$. By definition $x >_v y$ and $x =_v y$ with respect to an mgu σ', if $\mathrm{dep}(x\sigma') > \mathrm{dep}(y\sigma')$ and $\mathrm{dep}(x\sigma') = \mathrm{dep}(y\sigma')$, respectively.

3. Based on $>_v$ and $=_v$, the maximal variables in $\neg A_1 \vee \ldots \vee \neg A_n$ are called the *top variables*. The subset $\neg A_1, \ldots, \neg A_m$ of $\neg A_1, \ldots, \neg A_n$ ($m \leq n$) are the *top-variable literals* if each literal in $\neg A_1, \ldots, \neg A_m$ contains at least one of the top variables, and $\neg A_1 \vee \ldots \vee \neg A_m$ is the *top-variable subclause* of C.

The definitions of the top variable, the top-variable literal and the top-variable subclause are only in effect with respect to applications of the **Res** rule, to locate suitable **P-Res** eligible literals, therefore a top-variable resolution inference step can be seen as a special application of the **P-Res** rule. In general the top-variable technique does not requires one to select all the negative literals in the main premise C in an **Res** inference. In the PResT function, all the negative literals in C are selected, specifically for deciding satisfiability of the GQ clausal class.

The top-variable technique is devised to avoid term depth increase in the resolvents of GQ clauses. By **Algorithm 2**, $\mathrm{CompT}(C_1, \ldots, C_n, C)$ function takes a query clause $C = \neg A_1 \vee \ldots \vee \neg A_n \vee D$ as the main premise (in which all negative literals $\neg A_1 \vee \ldots \vee \neg A_n$ are selected), and GQ clauses $C_1 = B_1 \vee D_1, \ldots, C_n = B_n \vee D_n$ as the side premises. In the $\mathrm{CompT}(C_1, \ldots, C_n, C)$ function, 1. computes an mgu σ' such that $\sigma' = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$. In 2.–3., if a variable x in C is unified to be the deepest term $x\sigma'$ in $C\sigma'$, then x is the top variable. If $x\sigma'$ is a nested compound term, it may become a deeper term in the **Res** resolvent. To avoid this potential term depth increase, we compute a partial resolvent, by only resolving the top-variable literals of C with $C_1, \ldots, C_n$ in an **P-Res** inference. **Lemma 6.6** (in **Section 4.4**) formally states that in the application of the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{GQ}}$ refinement) to the GQ clauses, there is no term depth increase in the partial conclusions. Examples of applying the top-variable technique to GQ clauses (to avoid term depth increase in the resolvents) is given in **Section** 4.5 and **Section** 6.3. For readability, we sometimes call a **P-Res** inference endowed with the **T-Ref**$^{\mathsf{GQ}}$ refinement as the *top-variable resolution inference*.

The top-variable technique ensures to compute at least one top-variable literal with respect to a **Res** inference, formally stated as:

Lemma 4.4. *Suppose there is an application of the **Res** rule to $C_1, \ldots, C_n$ as the side premises and C as the main premise. Then the* $\mathrm{CompT}(C_1, \ldots, C_n, C)$ *function computes at least one top-variable literal in C.*

Proof. Since the **Res** rule is applicable to $C_1, \ldots, C_n$ and C, there exists an mgu σ' for $C_1, \ldots, C_n$ and C, therefore there exists at least one negative literal $\neg A\sigma'$ in $C\sigma'$ that is deeper than any other negative literals in $C\sigma'$. Hence, $\neg A$ is a top-variable literal (with respect to an **Res** inference to $C_1, \ldots, C_n$ and C). □

A similar claim to **Lemma** 4.4, for the 'MAXVAR' technique to decide satisfiability of the guarded clausal class with no term depth restrictions, can be found in [dNdR03, Page 45].

Although the **T-Ref**$^{\mathsf{GQ}}$ refinement is specially devised for deciding satisfiability of the GQ class, this refinement is also applicable to general first-order clauses, as the **T-Ref**$^{\mathsf{GQ}}$ refinement only uses admissible orderings with selection functions and a special application of the **P-Res** rule. By **Theorem** 4.2, we give the first main result of this paper.

Theorem 4.3. *The **T-Inf**$^{\mathsf{GQ}}$ system is sound and refutationally complete for general first-order clausal logic.*

In the resolution framework of [BG01], particularly in resolution-based decision procedures [FLHT01], the resolution and positive factoring rules are preferred to be applied with *a posteriori checking*. This checking means that in a resolution or factoring inference **I**, one first computes instantiations $C\sigma$ of the premise C (where σ is an mgu in **I**), and then determines the (strictly) maximal literal with respect to $C\sigma$ as the eligible literal. Opposite to a posteriori checking, *a prior checking* determines the (strictly) maximal literal with respect to the non-instantiated premise C as an eligible literal. Generally speaking, a posteriori checking is stronger than a priori checking, nonetheless, a posteriori checking requires one to pre-compute an mgu before finding the (strictly) maximal literals, which is not required when using a priori checking.

In the **Inf** system, we use a-posteriori checking, as shown in 2. in the **Fact** rule, and 1. and 2a. in the **P-Res** rule. However, thanks to the covering property of GQ clausal class, we can use a priori checking to avoid overheads pre-computing of unifications, caused by a posteriori checking. This property is briefly discussed in [GdN99] for guarded clauses, without providing proofs. We now formally prove this claim.

First we give a property of $\succ_{lpo}$ on covering clauses.

Lemma 4.5. *Let a covering clause C contain a compound-term literal L_1 and a non-compound-term literal L_2. Then $L_1 \succ_{lpo} L_2$.*

Proof. We distinguish two cases:

i) Suppose L_1 contains a ground compound term. By the covering property, C is ground. Then $L_1 \succ_{lpo} L_2$ as L_1 contains at least one function symbol but L_2 does not.

ii) Suppose L_1 contains a non-ground compound term t. By the covering property, $\text{var}(t) = \text{var}(L_1) = \text{var}(C)$. Since $\text{var}(L_2) \subseteq \text{var}(L_1)$ and L_1 contain at least one function symbol but L_2 does not, $L_1 \succ_{lpo} L_2$. ◻

By the **T-Ref$^{\textsf{GQ}}$** refinement and the covering property, if the (strictly) $\succ_{lpo}$-maximal literal with respect to a GQ clause C is literal L, then $L\sigma$ is the (strictly) $\succ_{lpo}$-maximal literal with respect to $C\sigma$, for any substitution σ. This means that the result of an application of a priori checking coincides with that of a posteriori checking with respect to the **T-Ref$^{\textsf{GQ}}$** refinement and GQ clauses. This is formally stated as:

Lemma 4.6. *Under the restrictions of the **T-Ref$^{\textsf{GQ}}$** refinement, in a GQ clause C, if an eligible literal L is (strictly) $\succ_{lpo}$-maximal with respect to C, then $L\sigma$ is (strictly) $\succ_{lpo}$-maximal with respect to $C\sigma$, for any substitution σ.*

Proof. In **Algorithm 1**, the $\text{Max}(C)$ function is used in either Lines 1–2 or 5–6.

The case in Lines 1–2 make the claim trivially holds, since C is ground. Lines 5–6 mean that C contains compound-term literals. By **Lemma 4.5**, the (strictly) $\succ_{lpo}$-maximal literal L in C is a compound-term literal. Since C is covering and L is compound-term literal, $\text{var}(L) = \text{var}(C)$. In C, suppose there is a literal L' that is distinct from L. By the facts that $\text{var}(L') \subseteq \text{var}(L)$ and $L \succ_{lpo} L'$ ($L \succeq_{lpo} L'$), $L\sigma \succ_{lpo} L'\sigma$ ($L\sigma \succeq_{lpo} L'\sigma$) under any substitution σ. Then $L\sigma$ is (strictly) $\succ_{lpo}$-maximal with respect to $C\sigma$. □

The property of **Lemma 4.6** can be easily generalised to any covering clause endowed with the idea of **T-Ref**$^{\mathsf{GQ}}$ refinement, since it is the covering property that makes the application of a priori checking possible.

By **Lemma 4.6**, from now on, we assume to use a priori checking to determine the (strictly) maximal literals in **Fact** and **P-Res** inferences. This also has the advantage in clearing the discussions and simplifying proofs related to the applications of these inference rules to guarded clauses.

4.4 Deciding the guarded clausal class

In this section, we show that the **T-Inf**$^{\mathsf{GQ}}$ system decides satisfiability of the guarded clausal class. Our goal is to show: given a finite signature $(\mathtt{C}, \mathtt{F}, \mathtt{P})$, applying the conclusion-deriving rules in the **T-Inf**$^{\mathsf{GQ}}$ system, namely the **Fact** and **P-Res** rules, to guarded clauses only derives guarded clauses that are of bounded depth and width using symbols in $(\mathtt{C}, \mathtt{F}, \mathtt{P})$.

By Lines 1–8 in **Algorithm 1**, no top-variable resolution inference is needed when premises are guarded clauses, therefore only a binary form of the **P-Res** rule is used in performing inference for guarded clauses. However in **Lemma 4.13** of this section, we investigate the case when performing the top-variable resolution inference on a flat clause and a set of guarded clauses, preparing us for understanding the inference between query clauses and guarded clauses. Note that although a guard is a negative flat literal, for readability we sometimes omit the negation symbol in front of guards.

In the **T-Inf**$^{\mathsf{GQ}}$ system, the **T-Ref**$^{\mathsf{GQ}}$ refinement ensures that any derived guarded clause is of bounded depth and width, which is achieved by restricting that in a guarded clause C, any eligible literal

i) shares the same variables set as C, and
ii) is the deepest literal in C.

The **T-Ref**$^{\mathsf{GQ}}$ refinement ensures the fact that given a guarded clause C, the eligible literal in C shares the same variable set as C, formally stated as:

Lemma 4.7. *Under the restrictions of the **T-Ref$^{\mathbf{GQ}}$** refinement, the eligible literal in a guarded clause C share the same variable set as C.*

Proof. By **Algorithm 1**, we distinguish three cases:

Lines 1–2: When C is ground the statement trivially holds.

Lines 3–6: Suppose C is a compound-term guarded clause and L is the eligible literal in C. By **Lemma 4.5** (if L is positive) and the definition of the SelectNC function (if L is negative), L is a compound-term literal. By the covering property, $\text{var}(L) = \text{var}(C)$.

Lines 7: Suppose C is a flat guarded clause and $\neg G$ is a guard in C. By 2. of **Definition 13**, $\text{var}(G) = \text{var}(C)$. □

The **T-Ref$^{\mathbf{GQ}}$** refinement also ensures that in a guarded clause, the deepest literal is eligible. In specific Lines 3–6 in **Algorithm 1** ensure that in a non-ground compound-term guarded clause, at least one of compound-term literals is eligible.

Next we look at how the restrictions of eligible literals ensure that applying the **T-Inf$^{\mathbf{GQ}}$** system to guarded clauses derives only clauses of bounded depth and width. We look into the unification for eligible literals in guarded clauses, starting with investigating the pairing property of compound-term eligible literals.

Lemma 4.8. *Let A_1 and A_2 be two simple and covering compound-term literals, and suppose A_1 and A_2 are unifiable using an mgu σ. Then compound terms in A_1 pair only compound terms in A_2, and vice-versa.*

Proof. We distinguish three cases:

i) The statement trivially holds when both A_1 and A_2 are ground.

ii) Suppose one of A_1 and A_2 is ground and the other one is non-ground. By the covering property, if a literal L contains a ground compound term, then L is ground. Hence, a non-ground compound term pairs either a ground compound term, or a constant. As it is impossible to unify a non-ground compound term and a constant, a non-ground compound term must pair a ground compound term.

iii) Suppose both A_1 and A_2 are non-ground. W.l.o.g. we represent A_1 and A_2 as $A_1(t, t', \ldots)$ and $A_2(u, u', \ldots)$, respectively. By the covering property and the assumption that A_1 and A_2 are non-ground, t, t', u and u' are non-ground compound terms, since the presence of ground compound terms means that a covering clause is ground.

Suppose t is a compound term. We prove that u is a compound term by contradiction. Assume that u is either a constant or a variable. Immediately u being a constant prevents the unification $t\sigma = u\sigma$. Now suppose u is a variable. As A_2 is a compound-term literal, w.l.o.g. we assume that u' is a compound term in A_2. Then t' is not a constant as it prevents the unification of u' and t', therefore t' is a variable or a compound term. We distinguish these two cases of t':

1. Suppose t' is a variable. By the covering property, w.l.o.g. we use $f(\ldots, x, \ldots)$, x, y and $g(\ldots, y, \ldots)$ to represent t, t', u and u', respectively. Then $A_1(t, t', \ldots)$ and $A_2(u, u', \ldots)$ are represented as $A_1(f(\ldots, x, \ldots), x, \ldots)$ and $A_2(y, g(\ldots, y, \ldots), \ldots)$, respectively. The unification between A_1 and A_2 is impossible.

2. Suppose t' is a compound term. By the covering property, w.l.o.g. we use $f(\overline{x})$, $g(\overline{x})$, y and $g(\ldots, y, \ldots)$ to represent t, t', u and u', respectively. Then $A_1(t, t', \ldots)$ and $A_2(u, u', \ldots)$ are represented as $A_1(f(\overline{x}), g(\overline{x}), \ldots)$ and $A_2(y, g(\ldots, y, \ldots), \ldots)$, respectively. Then there exists no unifier for A_1 and A_2.

Hence, u is a compound term. □

Let a guarded clause C be a premise in **Fact** or **P-Res** inferences. Then if guards $\neg G$ in C is not eligible literals, then the $\neg G$ literals will become the guard in the conclusion (after unification). This is formally stated as:

Lemma 4.9. *Let A_1 and A_2 be simple and covering atoms and suppose A_1 and A_2 are unifiable by an mgu σ. Further suppose G is a flat literals satisfying* $\mathrm{var}(A_1) = \mathrm{var}(G)$. *Then, if A_1 is a compound-term atom,* $\mathrm{var}(A_1\sigma) = \mathrm{var}(G\sigma)$ *and $G\sigma$ is a flat literal.*

Proof. Since $\mathrm{var}(A_1) = \mathrm{var}(G)$, it is immediate that $\mathrm{var}(A_1\sigma) = \mathrm{var}(G\sigma)$.

We prove that $G\sigma$ is flat by distinguish two cases of A_2:

i) Assume that A_2 is flat. This implies that σ substitutes variables in A_1 with either variables or constants. By the facts that G is flat and $\mathrm{var}(A_1) = \mathrm{var}(G)$, $G\sigma$ is flat.

ii) Assume that A_2 is a compound-term literal. By **Lemma** 4.8, compound terms in A_1 only pair compound terms in A_2. Then the mgu σ substitutes variables in A_1 with either variables or constants. By the facts that G is flat and $\mathrm{var}(A_1) = \mathrm{var}(G)$, $G\sigma$ is flat. □

Next, **Lemmas 4.10–4.11** consider non-guard literals occurring in conclusions. Lemma 4.6 in [GdN99] gives a similar result to **Lemma 4.10**, but a key 'covering' condition is missed.

First we look at the depth of eligible literals in conclusions.

Lemma 4.10 ([GdN99, Lemma 4.6])**.** *Suppose A_1 and A_2 are two simple and covering literals, and they are unifiable using an mgu σ. Then, $A_1\sigma$ is simple.*

Proof. If either of A_1 and A_2 is ground, or either of A_1 and A_2 is non-ground and flat, then immediately $A_1\sigma$ is simple.

Let both A_1 and A_2 be compound-term literals. By **Lemma 4.8**, the mgu σ substitutes variables in A_1 or A_2 with either constants or variables. By the fact that A_1 is simple, $A_1\sigma$ is simple. □

Next we look at the depth and width of non-eligible literals in conclusions.

Lemma 4.11. *Let A_1 and A_2 be two simple atoms satisfying* $\mathrm{var}(A_2) \subseteq \mathrm{var}(A_1)$. *Then given an arbitrary substitution σ, these properties hold:*

1. *If $A_1\sigma$ is simple, then $A_2\sigma$ is simple.*
2. $\mathrm{var}(A_2\sigma) \subseteq \mathrm{var}(A_1\sigma)$.

Further suppose that t and u are, respectively, compound terms occurring in A_1 and A_2, satisfying $\mathrm{var}(t) = \mathrm{var}(u) = \mathrm{var}(A_1)$. *Then* $\mathrm{var}(t\sigma) = \mathrm{var}(u\sigma) = \mathrm{var}(A_1\sigma)$.

Proof. By the assumptions that A_1 and $A_1\sigma$ are simple, σ does not cause term depth increase in $A_1\sigma$. Since $\mathrm{var}(A_2) \subseteq \mathrm{var}(A_1)$ and A_2 is simple, $A_2\sigma$ is simple.

By the facts that $\mathrm{var}(A_2) \subseteq \mathrm{var}(A_1)$ and $\mathrm{var}(t) = \mathrm{var}(u) = \mathrm{var}(A_1)$, immediately $\mathrm{var}(A_2\sigma) \subseteq \mathrm{var}(A_1\sigma)$ and $\mathrm{var}(t\sigma) = \mathrm{var}(u\sigma) = \mathrm{var}(A_1\sigma)$, respectively. □

Given a compound-term guarded clause C, one obtains a guarded clause by removing a compound-term literal from C, formally stated as:

Lemma 4.12. *Let $C = D \vee B$ be a guarded clause with B a compound-term literal. Let σ be a substitution that substitutes all variables in C with constants and variables. Then $D\sigma$ is a guarded clause.*

Proof. If σ is a ground substitution, then the lemma trivially holds. Let σ be a non-ground substitution. We prove that $D\sigma$ is simple, covering and contains a guard. Suppose G is a guard and t is a compound term in C. Since σ substitutes variables with either constants or variables, $D\sigma$ is simple, and $G\sigma$ is flat. Since $\text{var}(G) = \text{var}(C) = \text{var}(D)$, $\text{var}(G\sigma) = \text{var}(D\sigma)$. Then $G\sigma$ is a guard in $D\sigma$. Since $\text{var}(t) = \text{var}(C) = \text{var}(D)$, $\text{var}(t\sigma) = \text{var}(D\sigma)$. Hence, $D\sigma$ is covering. Then $D\sigma$ is a guarded clause. □

Next we give the properties of applying the top-variable resolution rule to a flat clause and guarded clauses.

Lemma 4.13. *In an application of the* **P-Res** *rule, endowed with the* **T-Ref$^{\mathbf{GQ}}$** *refinement, to a flat clause satisfying Line 9 of* **Algorithm 1** *(as the main premise) and guarded clauses (as the side premises), the following conditions hold.*

1. *In the main premise, top variables pair either constants or compound terms, and non-top variables pair constants and variables.*
2. *In the eligible literals of side premises, compound terms pair top variables, and either variables or constants pair non-top variables.*
3. *In the main premise, top variables x are unified with either constants or the compound term pairing x (modulo variables substituted with either variables or constants), and non-top variables are unified with constants and variables.*
4. *In the side premises, variables are unified with constants and variables.*
5. *Suppose a top variable x pairs a constant. Then in the main premise, all negative literals are the top-variable literals and all variables are unified with constants.*

Proof. It is assured that maximality is determined before the mgu is computed, as justified in **Lemma 4.6**. Thus, the **P-Res** rule (endowed with the **T-Ref$^{\mathbf{GQ}}$** refinement) is performed in the following form.

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

if the following conditions are satisfied.

1. No literal is selected in $D_1, \ldots, D_n$ and $B_1, \ldots, B_n$ are strictly $\succ_{lpo}$-maximal with respect to $D_1, \ldots, D_n$, respectively.

2a. If $n = 1$, i) either $\neg A_1$ is selected, or nothing is selected in $\neg A_1 \vee D$ and $\neg A_1$ is $\succ_{lpo}$-maximal with respect to D, and ii) $\sigma = \text{mgu}(A_1 \doteq B_1)$ or

2b. if $n > 1$ and there exists an mgu σ' such that $\sigma' = \text{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$, then $\sigma = \text{mgu}(A_1 \doteq B_1, \ldots, A_m \doteq B_m)$ where $m \leq n$.

3. All premises are variable disjoint.

Assume that the PResT function returns $\neg A_1 \vee \ldots \vee \neg A_m$ as top-variable literals. W.l.o.g. assume that $\neg A_t(\ldots, x, \ldots, y, \ldots)$ is a literal in $\neg A_1 \vee \ldots \vee \neg A_m$, and x is a top variable and y is a non-top variable (if it exists). Suppose $C_t = B_t(\ldots, t_1, \ldots, t_2, \ldots) \vee D_t$ is a side premise, in which t_1 and t_2 pair x and y, respectively.

1.: We show that t_1 is either a constant or a compound term and t_2 is either a constant or a variable. We distinguish two cases of C_t:

1.-1: Suppose C_t is ground. Then immediately t_1 is either a constant or a ground compound term. We prove that t_2 is a constant by contradiction. Assume that t_2 is not a constant, thus t_2 is a ground compound term. Hence, $\text{dep}(t_2) \geq \text{dep}(t_1)$. Since t_1 and t_2 are ground, $\text{dep}(t_2\sigma') \geq \text{dep}(t_1\sigma')$ with respect to the mgu σ'. Then $\text{dep}(y\sigma') \geq \text{dep}(x\sigma')$, which contradicts that y is non-top variable.

1.-2: Suppose C_t is not ground. By **Algorithm 1** and the covering property, C_t contains non-ground compound-term literals, otherwise at least one literal in C_t would be selected. By **Lemma 4.5**, the $>_{lpo}$-maximal literal B_t (with respect to D_t) is a compound-term literal. We prove that t_1 is a compound term and t_2 is a variable or a constant by contradiction. Assume t_1 is not a compound term. As B_t is a compound-term literal, suppose t is a compound term in B_t. W.l.o.g. suppose t pairs a variable z in A_t. By the facts that $\text{var}(t_1) \subseteq \text{var}(t)$ (due to the covering property) and $\text{dep}(t_1) < \text{dep}(t)$, $\text{dep}(t_1\sigma') < \text{dep}(t\sigma')$. Hence, $\text{dep}(x\sigma') < \text{dep}(z\sigma')$. This contradicts that x is a top variable. Thus t_1 must be a compound term. Now assume t_2 is neither a constant nor a variable, i.e., t_2 is a compound term. The facts that $\text{var}(t_1) = \text{var}(t_2)$ (by the covering property) and $\text{dep}(t_1) = \text{dep}(t_2)$ imply $\text{dep}(t_1\sigma') = \text{dep}(t_2\sigma')$. Hence $\text{dep}(x\sigma') = \text{dep}(y\sigma')$. This contradicts that y is not a top variable.

2.: Immediately follow *1.*.

3.: Because of the pairing property established in *1.*, the mgu σ substitutes top variables x with either constants or compound terms that x pairs (modulo variables substituted with either variables or constants), and substitutes any non-top variable y with either a constant or variable that y pairs.

4.: By *3.*.

5.: Suppose a top variable x pairs a constant. By the definition of the CompT function, for any non-top variable y, it is the case that $\text{dep}(x\sigma') > \text{dep}(y\sigma')$. The fact that x pairing a constant indicates that $x\sigma'$ is a constant, therefore $\text{dep}(x\sigma') = 0$. Then $\text{dep}(y\sigma') = 0$ and hence all variables in $\neg A_1 \vee \ldots \vee \neg A_n$ are top variables and are substituted with constants. □

Lemma 4.14. *In an application of the **P-Res** rule, endowed with the **T-Ref$^{\mathsf{GQ}}$** refinement, to a flat clause as the main premise and guarded clauses as the side premises, the **P-Res** resolvent is no deeper than its premises.*

Proof. By *3.–4.* in **Lemma 4.13** and the fact that the top-variable literals are resolved in a top-variable resolution inference. □

Now we investigate the applications of the **Fact** and **P-Res** rules to guarded clauses, starting with the application of the **Fact** rule.

Lemma 4.15. *In the application of the **Fact** rule (endowed with the **T-Ref$^{\mathsf{GQ}}$** refinement) to guarded clauses, the factors are guarded clauses.*

Proof. Consider a priori maximality checking revisit of the **Fact** rule (endowed with the **T-Ref$^{\mathsf{GQ}}$** refinement).

$$\textbf{Fact}: \quad \frac{C \vee A_1 \vee A_2}{(C \vee A_1)\sigma}$$

if the following conditions are satisfied.

1. Nothing is selected in $C \vee A_1 \vee A_2$.
2. A_1 is $\succ_{lpo}$-maximal with respect to C.
3. $\sigma = \text{mgu}(A_1 \doteq A_2)$.

Let the premise $C' = C \vee A_1 \vee A_2$ be a guarded clause. By **Algorithm 1**, we distinguish two cases of C':

Lines 1–2: By the fact that C' is simple and ground, the factor $(C \vee A_1)\sigma$ is also simple and ground, which is a guarded clause.

Lines 5–6: The premise C' is non-ground and contains positive compound-term literals. By **Lemma 4.5**, A_1 is a compound-term literal. By the covering property, $\mathrm{var}(A_2) \subseteq \mathrm{var}(A_1)$. Hence, A_2 must be a compound-term literal, otherwise A_1 and A_2 are not unifiable. Then by the covering property, $\mathrm{var}(A_2) = \mathrm{var}(A_1)$. By **Lemma 4.8**, compound terms in A_1 pair only compound terms in A_2, and vice-versa. Hence, the mgu σ substitutes variables with either variables or constants. By **Lemma 4.12**, the factor $(C \vee A_1)\sigma$ is a guarded clause. $\square$

Next, we discuss the resolvents of applying the **P-Res** rule to guarded clauses.

Lemma 4.16. *In the application of the* **P-Res** *rule (endowed with the* **T-Ref$^{\mathbf{GQ}}$** *refinement) to guarded clauses, the resolvents are guarded clauses.*

Proof. By **Algorithm 1**, we distinguish all possible cases of applying the **P-Res** rule to guarded clauses. In particular we consider the **P-Res** inferences when the top-variable technique is not used, since Line 9 in **Algorithm 1** requires a query clause as a premise. Let guarded clauses $C_1 = B_1 \vee D_1$ and $C = \neg A_1 \vee D$ be the positive and negative premises in an **P-Res** inference, deriving the resolvent $C' = (D_1 \vee D)\sigma$, where σ is the mgu of B_1 and A_1. By **Algorithm 1**, C is either ground, or contains a negative non-ground compound term literal or is a flat guarded clause (Lines 1–2, or 3–4 or 7–8, respectively), and C_1 satisfies either Lines 1–2 or 5–6. We distinguish three cases of C:

Lines 1–2: The negative premise C is ground. By the definition of guarded clauses, A_1 is either a ground flat literal or a ground compound-term literal. First suppose A_1 is a ground flat literal. Then the eligible literal B_1 of C_1 must be flat otherwise A_1 and B_1 are unifiable. By **Algorithm 1**, C_1 is a flat ground clause. Hence, it is immediate that the resolvent C' is a flat ground clause, that is, a guarded clause. Next assume that A_1 is a ground compound-term literal. Then B_1 is a compound-term literal, otherwise A_1 and B_1 are not unifiable. By **Lemma 4.8**, compound terms in A_1 pair only compound terms in B_1 and vice-versa. Then the mgu σ substitutes variables in B_1 with constants. By **Lemma 4.7**, all variables in C_1 are substituted with constants. Hence, C' is a ground and simple clause, that is, a guarded clause.

Lines 3–4: The negative premise C contains at least one negative non-ground compound-term literal. By **Algorithm 1**, A_1 is a negative compound-term literal, and C_1 is either i) a ground clause, or ii) contains positive non-ground compound-terms, but no negative non-ground compound-terms. By the facts that A_1 and B_1 are unifiable and **Lemma 4.5**, B_1 is a positive compound-term literal. Assume that G is a guard in C_1, L is a literal and t is a compound term in either C or C_1. As A_1 and B_1 satisfy conditions of **Lemma 4.9**, $G\sigma$ is flat and $\mathrm{var}(A_1\sigma) = \mathrm{var}(G\sigma)$. We know $\mathrm{var}(A_1\sigma) = \mathrm{var}(B_1\sigma)$. Then by **Lemma 4.7**, $\mathrm{var}(A_1\sigma) = \mathrm{var}(C\sigma)$ and $\mathrm{var}(B_1\sigma) = \mathrm{var}(C_1\sigma)$, therefore $\mathrm{var}(G\sigma) = \mathrm{var}(C_1\sigma) = \mathrm{var}(C_2\sigma) = \mathrm{var}(C')$. Hence $G\sigma$ is a guard of the resolvent C'. By **Lemma 4.7**, $\mathrm{var}(L) \subseteq \mathrm{var}(A_1)$ (or $\mathrm{var}(L) \subseteq \mathrm{var}(B_1)$). By **Lemma 4.10**, $A_1\sigma$ (or $B_1\sigma$) are simple. Then by 1. in **Lemma 4.11**, $L\sigma$ is simple. Hence, C' is simple. By **Lemma 4.7**, $\mathrm{var}(t) = \mathrm{var}(A_1) = \mathrm{var}(C_1)$ (or $\mathrm{var}(t) = \mathrm{var}(B_1) = \mathrm{var}(C_2)$). By **Lemma 4.11**, $\mathrm{var}(t\sigma) = \mathrm{var}(A_1\sigma)$ (or $\mathrm{var}(t\sigma) = \mathrm{var}(B_1\sigma)$). By the facts that $\mathrm{var}(A_1\sigma) = \mathrm{var}(C_1\sigma) = \mathrm{var}(C')$ (or $\mathrm{var}(B_1\sigma) = \mathrm{var}(C_2\sigma) = \mathrm{var}(C')$), $\mathrm{var}(t\sigma) = \mathrm{var}(C')$ and hence C' is covering. Then C is a guarded clause.

Line 7–8: The negative premise C is a flat guarded clause. By **Algorithm 1**, A_1 is a guard of C, and C_1 is either i) a ground clause, or ii) contains positive non-ground compound-terms, but no negative non-ground compound-terms. Suppose C_1 is ground. Then B_1 is either a ground flat literal, or a ground compound-term literal. In these cases, σ substitutes variables in A_1 with either constants or ground compound-terms of depth one. By **Definition 13**, $\mathrm{var}(C) = \mathrm{var}(A_1)$. Then σ substitutes variables in C with ground terms of depth less one. Hence, the resolvent C' is a simple and ground clause, namely a guarded clause. Next suppose C_1 contains positive non-ground compound-terms, but no negative non-ground compound-terms. Assume that G is a guard in C_1, L is a literal and t is a compound term in either C or C_1. As A_1 and B_1 satisfy conditions of **Lemma 4.9**, $G\sigma$ is flat and $\mathrm{var}(A_1\sigma) = \mathrm{var}(G\sigma)$. By the fact that $\mathrm{var}(A_1\sigma) = \mathrm{var}(B_1\sigma)$ and **Lemma 4.7**, $\mathrm{var}(A_1\sigma) = \mathrm{var}(C\sigma)$ and $\mathrm{var}(B_1\sigma) = \mathrm{var}(C_1\sigma)$, hence $\mathrm{var}(G\sigma) = \mathrm{var}(C_1\sigma) = \mathrm{var}(C_2\sigma) = \mathrm{var}(C')$. Then $G\sigma$ is a guard of the resolvent C'. By **Lemma 4.7**, $\mathrm{var}(L) \subseteq \mathrm{var}(A_1)$ (or $\mathrm{var}(L) \subseteq \mathrm{var}(B_1)$). By **Lemma 4.10**, $A_1\sigma$ (or $B_1\sigma$) are simple. Then by *1.* in **Lemma 4.11**, $L\sigma$ is simple. Hence C' is simple. By **Lemma 4.7**, $\mathrm{var}(t) = \mathrm{var}(A_1) = \mathrm{var}(C_1)$ (or $\mathrm{var}(t) = \mathrm{var}(B_1) = \mathrm{var}(C_2)$). By **Lemma 4.11**, $\mathrm{var}(t\sigma) = \mathrm{var}(A_1\sigma)$ (or $\mathrm{var}(t\sigma) = \mathrm{var}(B_1\sigma)$). By the facts that $\mathrm{var}(A_1\sigma) = \mathrm{var}(C_1\sigma) = \mathrm{var}(C')$ (or $\mathrm{var}(B_1\sigma) = \mathrm{var}(C_2\sigma) = \mathrm{var}(C')$), $\mathrm{var}(t\sigma) = \mathrm{var}(C')$ and hence C' is covering. Then C is a guarded clause. □

Lemmas 4.15–4.16 prove that applying the **Fact** and **P-Res** rules (endowed with the **T-Ref**$^{\mathbf{GQ}}$ refinement) to guarded clauses derive only guarded clauses. As guarded clauses are simple, these derived guarded clauses are of bounded depth. Let us now investigate the width of derived guarded clauses. Recall that by the width of a clause, we mean the number of distinct variables in that clause.

Lemma 4.17. *In applications of the* ***T-Inf***$^{\mathsf{GQ}}$ *system to guarded clauses, the derived guarded clause is no wider than at least one of its premises.*

Proof. By **Lemmas** 4.15–4.16, the conclusion of applying the **Fact** and **P-Res** rules to guarded clauses is a guarded clause. Then the guard in the conclusion contains all variables of this conclusion. In the conclusion of applying the **Fact** rule to guarded clauses, variables of the guard are inherited from that of a guard in the premise (modulo variable renaming and ground instantiation). In the conclusion of applying the **P-Res** rule to guarded clauses, variables of the guard are inherited from that of a guard in one of the positive premises (modulo variable renaming and ground instantiation). Hence in applying **T-Inf**$^{\mathsf{GQ}}$ system to guarded clauses, the conclusion is no wider than its (positive) premise. □

Now we give the first main result of this section.

Theorem 4.4. *The* ***T-Inf***$^{\mathsf{GQ}}$ *system decides satisfiability of the guarded clausal class.*

Proof. Suppose $(\mathtt{C}, \mathtt{F}, \mathtt{P})$ is a finite set of signature for the given guarded clauses. By **Lemmas** 4.15–4.16, applying the **T-Inf**$^{\mathsf{GQ}}$ system to guarded clauses derives the guarded clauses with bounded depth. By **Lemma** 4.17, the derived guarded clauses are of bounded width. These derived guarded clauses only use symbols in $(\mathtt{C}, \mathtt{F}, \mathtt{P})$, as no symbols are introduced in this derivation. □

4.5 Handling query clauses

In this section, we give our techniques to handle query clauses.

Suppose a GQ clausal set contains a query clause Q and a set N of guarded clauses. To handle Q, we first recursively apply two customised separation rules to replace Q by Horn guarded clauses (HG clauses). Suppose Q can be separated into HG clauses. Then by **Theorem** 6.4, the **T-Inf**$^{\mathsf{GQ}}$ system decides satisfiability of $Q \cup N$. If Q cannot be expressed in HG clauses, we the apply the top-variable resolution rule to Q (as a main premise) and clauses in N (as side premises), deriving the top-variable resolvent R. This top-variable resolvent R is not necessarily a GQ clause, hence in the last step a form of structural transformation is applied to R to replace it by an equisatisfiable set of GQ clauses.

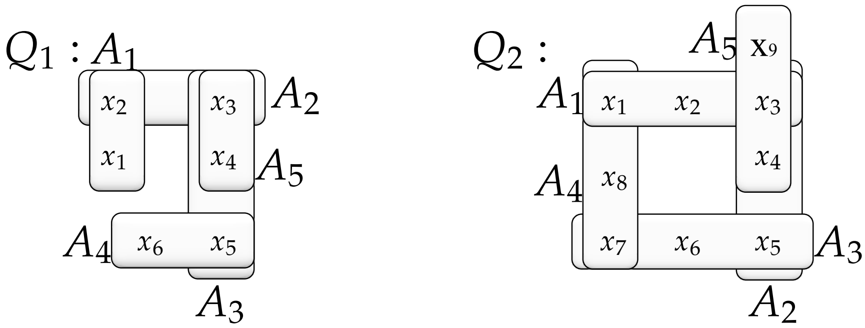

Figure 4.1: The hypergraphs associated with Q_1 and Q_2

Basic notions of query clauses

To analyse query clauses, we introduce the notions *surface literal*, *chained variables* and *isolated variables* with respect to query clauses.

Definition 15. *Let Q be a query clause. Then in Q, a literal L is a* surface literal (with respect to Q) *if there exists no literal L' such that* $\mathrm{var}(L) \subset \mathrm{var}(L')$.

Suppose in Q, L_1 and L_2 are two surface literals such that $\mathrm{var}(L_1) \neq \mathrm{var}(L_2)$. *Then x is a* chained variable (with respect to Q) *if x occurs in* $\mathrm{var}(L_1) \cap \mathrm{var}(L_2)$. *The other non-chained variables in Q, are* isolated variables (with respect to Q).

For example, in

$$Q_1 = \neg A_1(x_1, x_2) \vee \neg A_2(x_2, x_3) \vee \neg A_3(x_3, x_4, x_5) \vee \neg A_4(x_5, x_6) \vee \neg A_5(x_3, x_4),$$

$\neg A_1(x_1, x_2)$, $\neg A_2(x_2, x_3)$, $\neg A_3(x_3, x_4, x_5)$ and $\neg A_4(x_5, x_6)$ are surface literals, but $\neg A_5(x_3, x_4)$ is not as $\mathrm{var}(A_5) \subset \mathrm{var}(A_3)$. Then, with respect to Q_1, x_2, x_3, x_5 are chained variables and x_1, x_4, x_6 are isolated variables. In

$$\begin{aligned} Q_2 = \neg A_1(x_1, x_2, x_3) \vee \neg A_2(x_3, x_4, x_5) \vee \neg A_3(x_5, x_6, x_7) \vee \\ \neg A_4(x_1, x_7, x_8) \vee \neg A_5(x_3, x_4, x_9), \end{aligned}$$

all literals are surface literals, therefore with respect to Q_2, x_1, x_3, x_4, x_5, x_7 are chained variables and x_2, x_6, x_8, x_9 are isolated variables. Figure 4.1 shows the associated hypergraphs with Q_1 and Q_2.

Using **Definition 15**, we define two special forms of query clauses.

Definition 16. *A* chained-only query clause (CO) *and an* isolated-only query clause (IO) *are query clauses containing only chained variables, and only isolated*

Figure 4.2: The hypergraphs associated with Q_3 and Q_4

variables, respectively.

For example, $Q_3 = \neg A(x_1, x_2) \vee \neg A_2(x_2, x_3, x_4) \vee \neg A_3(x_1, x_3, x_4)$ is a CO clause and $Q_4 = \neg A_1(x_1) \vee \neg A_2(x_1, x_2) \vee \neg A_3(x_1, x_2, x_3)$ is an IO clause. Figure 4.2 shows the associated hypergraphs with Q_3 and Q_4.

The customised separation rules

In this section, by our customised notions of query clauses, we present two novel separations rules. These rules are variations of the **Sep** rule, and they provide goal-oriented approaches to separate query clauses. We then formally prove that these variations can be used as simplification rules in the **T-Inf**$^{\mathsf{GQ}}$ system.

Recall that a clause is *decomposable* if this clause consists of variable-disjoint subclauses, otherwise this clause is *indecomposable*.

A decomposable query clause is separated by

The QuerySepOne rule

$$\frac{N \cup \{C \vee D\}}{N \cup \{C \vee \neg p_1, \neg p_2 \vee D, p_1 \vee p_2\}}$$

if the following conditions are satisfied.

1. $C \vee D$ is a decomposable query clause.
2. C and D are not empty.
3. $\mathrm{var}(C) \cap \mathrm{var}(D) = \emptyset$.
4. Propositional variables p_1 and p_2 do not occur in $N \cup \{C \vee D\}$.

An indecomposable query clause is separated using

The QuerySepTwo rule

$$\frac{N \cup \{C \vee L(\overline{x}, \overline{y}) \vee D\}}{N \cup \{C \vee L(\overline{x}, \overline{y}) \vee P(\overline{x}), \neg P(\overline{x}) \vee D\}}$$

if the following conditions are satisfied.

1. $C \vee L(\overline{x}, \overline{y}) \vee D$ is an indecomposable query clause.
2. $L(\overline{x}, \overline{y})$ is a surface literal and $\text{var}(C) \subseteq \text{var}(L)$.
3. $\overline{x}$ are chained variables and $\overline{x} \subseteq \text{var}(D)$.
4. $\overline{y}$ are isolated variables and $\overline{y} \cap \text{var}(D) = \emptyset$.
5. Predicate symbol P does not occur in $N \cup \{C \vee L(\overline{x}, \overline{y}) \vee D\}$.

Next we prove that the **QuerySepOne** and **QuerySepTwo** rules are variations of the **Sep** rule. The **QuerySepOne** rule is immediately a variation the **Sep** rule. The fact that the **QuerySepTwo** rule is a variation of the **Sep** rule is formally stated as:

Lemma 4.18. *Given a clausal set N, the following conditions are satisfied.*

1. *If the **QuerySepTwo** rule is applicable to N, then, the **Sep** rule is applicable to N.*
2. *Applying the **QuerySepTwo** and **Sep** rules to N, respectively, derive the same conclusions.*

Proof. Suppose $N = N' \cup \{C \vee L(\overline{x}, \overline{y}) \vee D\}$ is the **QuerySepTwo** premises. We aim to prove that the **Sep** rule is applicable to N, and applying the **QuerySepTwo** and **Sep** rules to N, respectively, derive exactly the same conclusions.

1.: We aim to prove that $C \vee L(\overline{x}, \overline{y}) \vee D$ has the following property: 1) $\text{var}(C \vee L(\overline{x}, \overline{y})) \nsubseteq \text{var}(D)$, 2) $\text{var}(D) \nsubseteq \text{var}(C \vee L(\overline{x}, \overline{y}))$ and 3) $C \vee L(\overline{x}, \overline{y}))$ and D are not empty. By 3. in the **QuerySepTwo** rule, 3) trivial holds. Now we prove 1)–2). By 4. in the **QuerySepTwo** rule, $\overline{y} \cap \text{var}(D) = \emptyset$. This implies $\text{var}(C \vee L(\overline{x}, \overline{y})) \nsubseteq \text{var}(D)$. We prove $\text{var}(D) \nsubseteq \text{var}(C \vee L(\overline{x}, \overline{y}))$ by contradiction. Suppose $\text{var}(D) \subseteq \text{var}(C \vee L(\overline{x}, \overline{y}))$. By 2. of the **QuerySepTwo** rule, $\text{var}(C) \subseteq \text{var}(A)$. Then $\text{var}(C \vee L(\overline{x}, \overline{y}) \vee D) = \{\overline{x}, \overline{y}\}$. This contradicts 3. in the **QuerySepTwo** rule that $\overline{x}$ are chained variables.

2.: By 3.–5. in the **QuerySepTwo** rule, $\overline{x} = \text{var}(C \vee L(\overline{x}, \overline{y})) \cap \text{var}(D)$. Hence, the **Sep** rule can separate $C \vee L(\overline{x}, \overline{y}) \vee D$ into $C \vee L(\overline{x}, \overline{y}) \vee P(\overline{x})$ and $\neg P(\overline{x}) \vee D$ where $\overline{x} = \text{var}(C \vee L(\overline{x}, \overline{y})) \cap \text{var}(D)$. □

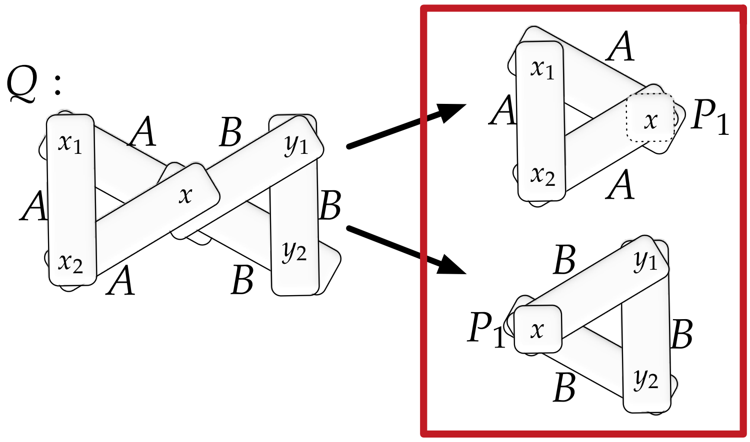

Figure 4.3: The application of the **Sep** rule to Q

Indeed the **Sep** rule is more powerful than the **QuerySepOne** and **QuerySepTwo** rules. Given a query clause

$$Q = \neg A(x_1, x) \vee \neg A(x_1, x_2) \vee \neg A(x_2, x) \vee \neg B(y_1, x) \vee \neg B(y_1, y_2) \vee \neg B(y_2, x),$$

the **Sep** rule separates it into an HG clause

$$\neg A(x_1, x) \vee \neg A(x_1, x_2) \vee \neg A(x_2, x) \vee P(x)$$

and a query clause $\neg B(y_1, x) \vee \neg B(y_1, y_2) \vee \neg B(y_2, x) \vee \neg P(x)$ where P is a new predicate symbol. Yet neither **QuerySepOne** nor **QuerySepTwo** is applicable to Q, since Q is an indecomposable CO clause. Figure 4.3 on the next page shows the process of applying the **Sep** rule to Q and the derived clauses are in the coloured box.

The **QuerySepOne** and **QuerySepTwo** rules are specially devised for separating query clauses. Unlike the **Sep** rule, the **QuerySepOne** and **QuerySepTwo** rules specifically use our notions for query clauses. This is due to the fact that identifying the conclusions of applying the **Sep** rule to query clauses is difficult. Moreover in the **QuerySepOne** and **QuerySepTwo** conclusions, the polarity of newly introduced symbols are assigned in a way such that these conclusions are in our desire form, namely the GQ clauses. For example in conclusions of the **QuerySepOne** rule, we use not one, but two propositional variables, so that by our assigning of polarity to fresh propositional variables p_1 and p_2, applying the **QuerySepOne** rule to a decomposable query clause derives two query clauses $C \vee \neg p_1$ and $\neg p_2 \vee D$ and a guarded clause $p_1 \vee p_2$.

The **QuerySepOne** and **QuerySepTwo** rules are sound, formally stated as:

Lemma 4.19. ***QuerySepOne*** *and* ***QuerySepTwo*** *preserve logical equivalence.*

Proof. By **Lemmas** 3.4–4.18, the **QuerySepTwo** rule preserves logical equivalence. Immediately, the statement holds for the **QuerySepOne** rule. □

Note that in the applications of the **QuerySepOne** and **QuerySepTwo** rules, the newly introduced predicate symbols are smaller than those in **QuerySepOne** and **QuerySepTwo** premises, respectively. Hence, as long as the applications of these separation rules do not introduce infinitely many predicate symbols, one can regard the **QuerySepOne** and **QuerySepTwo** rules as simplification rules in the resolution framework of [BG01]. **Lemma** 4.27 (in **Section** 4.6) formally proves that in our procedures, these separation rules only introduce finitely many predicate symbols when separating query clauses. Thus we consider the **QuerySepOne** and **QuerySepTwo** rules as simplification rules for extending the **T-Inf**$^{\mathsf{GQ}}$ system.

Separating query clauses

In this section, we investigate the applications of the **QuerySepOne** and **QuerySepTwo** rules to query clauses. We start with the **QuerySepOne** rule.

Lemma 4.20. *Suppose Q is a decomposable query clause. Then recursively applying the* ***QuerySepOne*** *rule to Q separate it into less wide indecomposable query clauses and* HG *clauses.*

Proof. By the definitions of indecomposable query clauses and HG clauses. □

By 2.–4. in the **QuerySepTwo** rule, one can apply the **QuerySepTwo** rule to indecomposable query clause Q only if there exists a surface literal in Q where both chained and isolated variables occur. Based on this fact, we look at how the **QuerySepTwo** rule is applied to indecomposable query clauses.

Lemma 4.21. *Suppose Q is an indecomposable query clause, in which a surface literal that contains both chained and isolated variables occurs. Then the* ***QuerySepTwo*** *rule separates Q into less wide query clauses and* HG *clauses.*

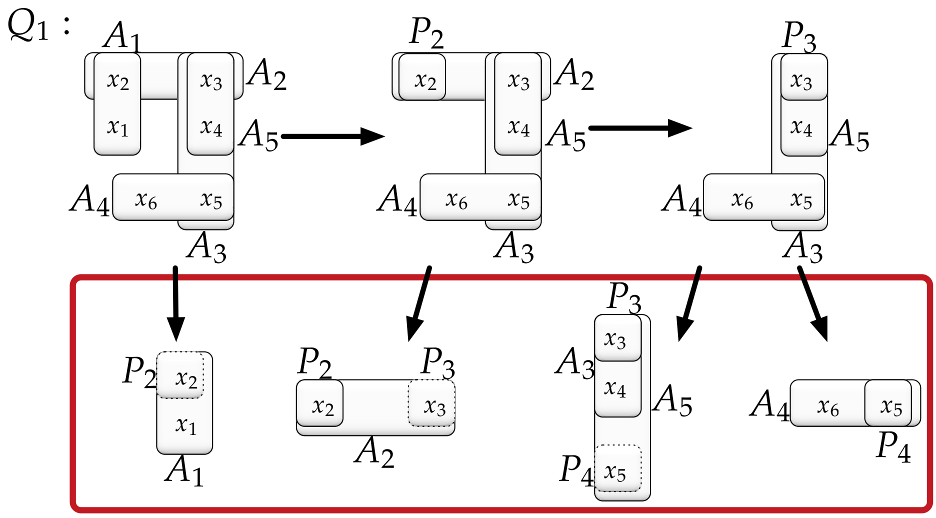

Figure 4.4: Separating Q_1 into HG clauses

Proof. Suppose $C \vee L(\overline{x}, \overline{y}) \vee D$ is an indecomposable query clause as the main premise of the **QuerySepTwo** rule. Further suppose applying the **QuerySepTwo** rule to $C \vee L(\overline{x}, \overline{y}) \vee D$ derives $\neg P(\overline{x}) \vee D$ and $C \vee L(\overline{x}, \overline{y}) \vee P(\overline{x})$.

First consider $\neg P(\overline{x}) \vee D$. As D is a query clause, $\neg P(\overline{x}) \vee D$ is a query clause. By the facts that all variables in $\neg P(\overline{x}) \vee D$ occur in $C \vee L(\overline{x}, \overline{y}) \vee D$, but $\neg P(\overline{x}) \vee D$ does not contain $\overline{y}$, $\neg P(\overline{x}) \vee D$ is less wide than $C \vee L(\overline{x}, \overline{y}) \vee D$.

Next consider $C \vee L(\overline{x}, \overline{y}) \vee P(\overline{x})$. By 2. in the **QuerySepTwo** rule, the variables in $L(\overline{x}, \overline{y})$ are the same as the variables in $C \vee L(\overline{x}, \overline{y}) \vee P(\overline{x})$. Then $C \vee L(\overline{x}, \overline{y}) \vee P(\overline{x})$ is flat with a guard $L(\overline{x}, \overline{y})$. It is an HG clause. We prove that $C \vee L(\overline{x}, \overline{y}) \vee P(\overline{x})$ is less wide than $C \vee L(\overline{x}, \overline{y}) \vee D$ by contradiction. Suppose $\mathrm{var}(D) \subseteq \overline{x} \cup \overline{y}$. By 2. in the **QuerySepTwo** rule, $\mathrm{var}(D \vee C) \subseteq \mathrm{var}(L)$. This contradicts that $\overline{x}$ are chained variables. Hence, D contains more types of variables than $\overline{x} \cup \overline{y}$. □

By **Lemmas** 4.20–4.21, applying the **QuerySepOne** and **QuerySepTwo** rules to a query clause derives a new query clause, therefore one can recursively apply these separation rules to a query clause. We use **Q-Sep** to denote this procedure. For example, applying the **Q-Sep** procedure to

$$Q_1 = \neg A_1(x_1, x_2) \vee \neg A_2(x_2, x_3) \vee \neg A_3(x_3, x_4, x_5) \vee \neg A_4(x_5, x_6) \vee \neg A_5(x_3, x_4),$$

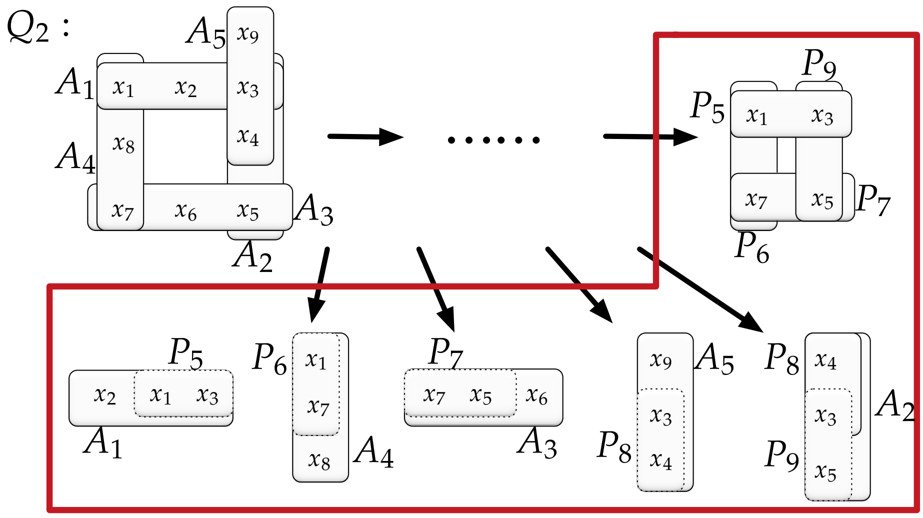

Figure 4.5: Separates Q_2 into HG clauses and an indecomposable CO clause

derives HG clauses:

$$\neg A_1(x_1, x_2) \vee P_2(x_2), \qquad \neg A_3(x_3, x_4) \vee \neg A_5(x_3, x_4, x_5) \vee \neg P_4(x_3) \vee P_3(x_5),$$
$$\neg A_4(x_5, x_6) \vee \neg P_4(x_5), \qquad \neg A_2(x_2, x_3) \vee \neg P_2(x_2) \vee P_3(x_3).$$

The **Q-Sep** procedure separates

$$Q_2 = \neg A_1(x_1, x_2, x_3) \vee \neg A_2(x_3, x_4, x_5) \vee \neg A_3(x_5, x_6, x_7) \vee$$
$$\neg A_4(x_1, x_7, x_8) \vee \neg A_5(x_3, x_4, x_9),$$

into HG clauses:

$$\neg A(x_1, x_2, x_3) \vee P_5(x_1, x_3), \qquad D(x_1, x_7, x_8) \vee P_6(x_1, x_7),$$
$$C(x_5, x_6, x_7) \vee P_7(x_5, x_7), \qquad \neg E(x_3, x_4, x_9) \vee P_8(x_3, x_4),$$
$$\neg B(x_3, x_4, x_5) \vee \neg P_8(x_3, x_4) \vee P_9(x_3, x_5)$$

and a CO clause $\neg P_5(x_1, x_3) \vee \neg P_9(x_3, x_5) \vee \neg P_7(x_5, x_7) \vee \neg P_6(x_1, x_7)$.

Figures 4.4 and 4.5 show how the **Q-Sep** procedure separates Q_1 into HG clauses, and separates Q_2 into a CO clause and HG clauses, respectively. The produced clauses are framed in the coloured box.

The application of **Q-Sep** to a query clause terminates if the derived (or given) query clause Q is indecomposable and contains either only chained variables or only isolated variables, namely an indecomposable CO clause or an indecomposable IO clause, respectively.

Analysis of indecomposable IO clauses and HG clauses reveals the following property:

Lemma 4.22. *An indecomposable IO clause is an HG clause.*

Proof. Suppose Q is an IO clause. Recall that if Q contains two surface literals L_1 and L_2 such that $\mathrm{var}(L_1) \neq \mathrm{var}(L_2)$ and $x \in \mathrm{var}(L_1) \cap \mathrm{var}(L_2)$, then x is a chained variable with respect to Q. Since Q contains no chained variables, it is the case that either i) Q contains only one surface literal, or ii) Q contains multiple surface literal and each pair L_1 and L_2 of surface literals satisfies either $\mathrm{var}(L_1) = \mathrm{var}(L_2)$ or $\mathrm{var}(L_1) \cap \mathrm{var}(L_2) = \emptyset$. We distinguish these two cases:

i): An indecomposable IO clause Q is flat, negative and contains only one surface literal L. By the definition of surface literal, $\mathrm{var}(L) = \mathrm{var}(Q)$. Hence, Q is an HG clause with a guard L.

ii): If in Q, any pair L_1 and L_2 of surface literals satisfies $\mathrm{var}(L_1) = \mathrm{var}(L_2)$, then it is the same case as i), except that there are guards L_1 and L_2. If there exists a pair L_1 and L_2 of surface literals satisfies $\mathrm{var}(L_i) \cap \mathrm{var}(L_j) = \emptyset$, then Q is decomposable. This contradicts the assumption. □

Next we give the result of applying the **Q-Sep** procedure to query clauses.

Lemma 4.23. *Applying the* **Q-Sep** *procedure to a query clause replaces that query clause by less wide HG clauses (and an indecomposable CO clause).*

Proof. By **Lemmas** 4.20–4.22. □

Note that depending on surface literals one picks, applying the **Q-Sep** procedure to a query clause may derive different sets of HG clauses (and an indecomposable CO clause).

If we consider a query clause as a hypergraph, then the **Q-Sep** procedure 'cuts the branches off' the hypergraph. Interestingly, the **Q-Sep** procedure handles query clauses similarly as the so-called *GYO-reduction* in [YO79]. Using the notion of cyclic queries in [BFMY83], GYO-reduction identifies cyclic conjunctive queries by recursively removing branches ('ears') in the hypergraph of queries, and it reduces a conjunctive query to an empty formula if the query is acyclic. In our definition of query clauses, an 'ear' map to the surface literal containing both isolated and chained variables, and by the **Q-Sep** procedure, these surface literals are removed from query clauses. Hence, one can regard the **Q-Sep** procedure (to query clauses) as an implementation of GYO-reduction (to conjunctive queries). The fact that an acyclic conjunctive query can be expressed as guarded formulas is also given in [FFG02, GLS03].

Cyclicity of a query clause can be checked by applying the **Q-Sep** procedure to it. This is formally stated as:

Lemma 4.24. *Applying the* **Q-Sep** *procedure to a query clause Q replaces it by*

- *HG clauses if Q is acyclic,*
- *HG clauses and an indecomposable CO clause if Q is cyclic.*

Proof. By the definition of GYO-reduction in [YO79]. □

By **Lemma 4.24** and the facts the **Q-Sep** procedure separates Q_1 into HG clauses and separates Q_2 into HG clauses and an indecomposable CO clause, Q_1 and Q_2 are identified as an acyclic query and a cyclic query, respectively.

By **Theorem 4.4**, the **T-Inf**$^{\mathsf{GQ}}$ system decides the guarded clausal class. By **Lemma 4.23**, query clauses can be replaced by an equisatisfiable set of *HG clauses* and an *indecomposable CO clause*. Hence, the only new class of clauses that we cannot handle are indecomposable CO clauses. In the next section, we give techniques to handle these clauses.

Handling indecomposable CO clauses

In this section, we first show how the top-variable resolution rule solves the term depth increase problem in reasoning with indecomposable CO clauses and guarded clauses. As these top-variable resolvents are not necessarily in the GQ clausal class, we devise a novel form of structure transformation to handle these resolvents. For readability, in the following sections we sometimes refer indecomposable CO clauses as CO clauses.

In an indecomposable CO clause such as

$$Q_3 = \neg P_5(x_1, x_3) \lor \neg P_9(x_3, x_5) \lor \neg P_7(x_5, x_7) \lor \neg P_6(x_1, x_7),$$

variable x_1, x_3, x_5, x_7 forms a 'cycle' through literals P_5, P_9, P_7, P_6, as shown in the top-right corner of Figure 4.5. If one applies the **Res** rule, or the binary **Res** rule, to Q_3 and guarded clauses, nested compound-terms may occur in the conclusions. For example, consider a GQ clausal set N containing Q_3 and guarded clauses:

$$\begin{aligned}
C_1 &= P_5(x, g(x, y, z_1, z_2))^* \lor \neg G_1(x, y, z_1, z_2),\\
C_2 &= \neg G_2(x, y, z_1, z_2) \lor P_9(g(x, y, z_1, z_2), x)^* \lor A(h(x, y, z_1, z_2)),\\
C_3 &= P_7(f(x), x)^* \lor \neg G_3(x),\\
C_4 &= P_6(f(x), x)^* \lor \neg G_4(x).
\end{aligned}$$

Suppose one applies the **Res** rule to $C_1, \ldots, C_4$ as the side premises and Q_3 (with all negative literals selected) as the main premise, deriving the resolvent:

$$\begin{aligned}
R_1 = {}&\neg G_3(x) \lor \neg G_4(x) \lor\\
&\neg G_1(f(x), y, z_1, z_2) \lor \neg G_2(f(x), y, z_1, z_2) \lor A(h(f(x), y, z_1, z_2)).
\end{aligned}$$

A nested compound-term literal $A(h(f(x), y, z_1, z_2))$ occurs in the resolvent R_1. Next, suppose one applies the binary **Res** rule to clauses in N. Applying the binary **Res** rule to C_3 and Q_3 (with $\neg P_7(x_5, x_7)$ selected) derives

$$R_2 = \neg P_5(x_1, x_3) \lor \neg P_9(x_3, f(x)) \lor \neg G_3(x) \lor \neg P_6(x_1, x).$$

Then apply the binary **Res** rule to C_2 and R_2 (with $\neg P_9(x_3, f(x))$ selected) derives

$$\begin{aligned}
R_3 = {}&\neg P_5(x_1, x_3) \lor\\
&\neg G_3(x) \lor \neg P_6(x_1, x) \lor \neg G_2(f(x), y, z_1, z_2) \lor A(h(f(x), y, z_1, z_2)),
\end{aligned}$$

in which, again, the nested compound-term literal $A(h(f(x), y, z_1, z_2))$ occurs.

Now we show how the top-variable technique tackles this term depth increase problem. By **Algorithms 1–2**, the top-variable resolution rule is applied to Q_3 and $C_1 \ldots, C_4$ as follows.

1. The PResT(Q_3, N) function first selects all negative literals in Q_3, and then finds the **Res** side premises of Q_3, namely $C_1, \ldots, C_4$.

2. The mgu of $C_1, \ldots, C_4$ and Q_3 is

$$\{x_1 \mapsto f(x), x_5 \mapsto f(x), x_7 \mapsto x, x_3 \mapsto g(f(x), y, z_1, z_2)\}$$

for variables in Q_3, hence, x_3 is the top variable.

3. PResT(Q_3, N) then returns $\neg P_5(x_1, x_3)$ and $\neg P_9(x_3, x_5)$ as top-variable literals. An top-variable resolution inference is performed on Q_3, C_1 and C_2, deriving the top-variable resolvent

$$\begin{aligned}R = \neg G_1(x, y, z_1, z_2) \vee \neg G_2(x, y, z_1, z_2) \vee \\ A(h(x, y, z_1, z_2))^* \vee \neg P_7(x, x_7) \vee \neg P_6(x, x_7),\end{aligned}$$

which does not contain any nested compound terms.

4. There is no possible inference for clauses in $N \cup R$, hence $N \cup R$ is saturated.

Although the top-variable resolvent R does not contain nested compound terms, R is wider than any of its premise $C_1, \ldots, C_4, Q_3$; moreover it is even not a GQ clause. Using a new predicate symbol P_8 (and a respective literal $\neg P_8(x, y, z_1, z_2)$) to define $\neg G_1(x, y, z_1, z_2) \vee \neg G_2(x, y, z_1, z_2) \vee A(h(x, y, z_1, z_2))$, the top-variable resolvent R is replaced by its equisatisfiable set of GQ clauses:

$$\begin{aligned}C_5 &= \neg G_1(x, y, z_1, z_2) \vee \neg G_2(x, y, z_1, z_2) \vee A(h(x, y, z_1, z_2))^* \vee P_8(x, y, z_1, z_2), \\ Q_4 &= \neg P_7(x, x_7) \vee \neg P_6(x, x_7) \vee \neg P_8(x, y, z_1, z_2).\end{aligned}$$

Note that C_5 is a guarded clause and Q_4 is a query clause. Since Q_4 is an indecomposable query clause, we can apply the **Q-Sep** procedure to it (using a new predicate symbol P_{10}), to replace it by HG clauses:

$$\begin{aligned}C_6 &= \neg P_7(x, x_7) \vee \neg P_6(x, x_7) \vee \neg P_{10}(x), \\ C_7 &= \neg P_8(x, y, z_1, z_2) \vee P_{10}(x).\end{aligned}$$

Figure 4.6 shows how the **Q-Sep** procedure separates Q_4 into Horn guarded clauses C_6 and C_7, and The produced clauses are framed in the coloured box. Then the top-variable resolvent R is replaced by guarded clauses C_5, C_6 and C_7. To sum up, given an GQ clausal set $\{Q_3, C_1, \ldots, C_4\}$, the $\mathbf{Inf}^{\mathsf{GQ}}$ system derives a saturated GQ clausal set $\{Q_3, C_1, \ldots, C_7\}$.

Transforming the top-variable resolvent (of an indecomposable CO clause and a guarded clausal set) to GQ clauses is not straightforward. We use notions *connected top variables* and *closed top-variable subclauses* to find disjunctively connected GQ subclauses in the top-variable resolvents.

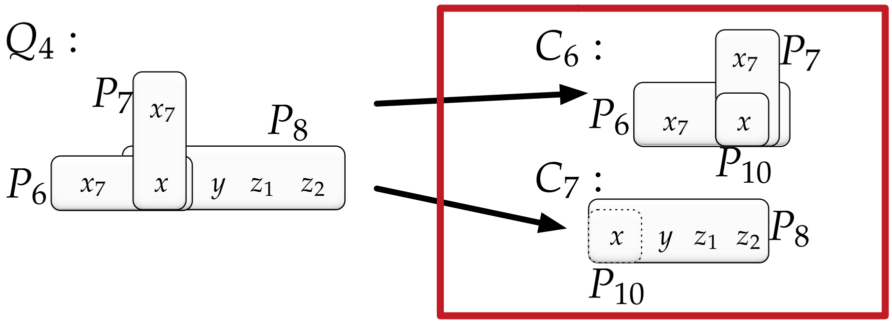

Figure 4.6: Separating Q_4 into HG clauses C_6 and C_7

Definition 17. *In a* **P-Res** *inference step* **I** *(endowed with the* **T-Ref**$^{\mathsf{GQ}}$ *refinement) to an indecomposable* CO *clause with the top-variable subclause* C*, and guarded clauses, we say that*

1. *in* C*, top variables* x_i *and* x_j *are* connected *(with respect to* **I***) if there exists a sequence of top variables* $x_i, \ldots, x_j$ *such that each pair of adjacent variables co-occurs in a top-variable literal, and*
2. *subclause* C' *is a* closed top-variable subclause *of* C *(with respect to* **I***) if*
   (a) *each pair of top variables in* C' *are connected, and*
   (b) *top variables in* C' *do not connect to top variables in* C*, but not in* C'*.*

Suppose a top-variable resolution inference is performed on an indecomposable CO clause Q and a guarded clausal set N. Then each closed top-variable subclause in Q is resolved with a subset N' of N, and the disjunction of remainders of N' forms a guarded clause (after unification). Consider the previous example. In a top-variable resolution inference of $C_1, \ldots, C_4$ and Q_3, $\neg P_5(x_1, x_3) \vee \neg P_9(x_3, x_5)$ is the top-variable subclause, which is also the only closed top-variable subclause, as x_3 is the only top variable. The matching side premises of $\neg P_5(x_1, x_3) \vee \neg P_9(x_3, x_5)$ are C_1 and C_2. Then the disjunction of remainders (after unification) of C_1 and C_2 forms a guarded clause $C' = \neg G_1(x, y, z_1, z_2) \vee \neg G_2(x, y, z_1, z_2) \vee A(h(x, y, z_1, z_2))$ in the top-variable resolvent $R = \neg G_1(x, y, z_1, z_2) \vee \neg G_2(x, y, z_1, z_2) \vee A(h(x, y, z_1, z_2))^* \vee \neg P_7(x, x_7) \vee \neg P_6(x, x_7)$. In the previous example, to abstract C' from R, we use a fresh predicate symbol P_8 to transform R into GQ clauses Q_4 and C_5.

The top-variable resolvents are handled by

The T-Trans rule

In an application of the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{GQ}}$ refinement) to an indecomposable CO clause $\neg A_1 \vee \ldots \vee \neg A_n$ with the top-variable subclause $\neg A_1 \vee \ldots \vee \neg A_m$ where $m \leq n$, and guarded clauses $C_1 = B_1 \vee D_1, \ldots, C_n = B_n \vee D_n$, the top variable resolvent is $R = (\neg A_{m+1} \vee \ldots \vee \neg A_n \vee D_1 \vee \ldots \vee D_m)\sigma$ where σ is an mgu such that $\sigma = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_m \doteq B_m)$.
Suppose $\neg A_1 \vee \ldots \vee \neg A_m$ is partitioned into closed top-variable subclauses $C'_1, \ldots, C'_t$, so that R is represented as $(\neg A_{m+1} \vee \ldots \vee \neg A_n \vee D'_1 \vee \ldots D'_t)\sigma$. Then the top-variable resolvent R is transformed using

$$\frac{N \cup \{(\neg A_{m+1} \vee \ldots \neg A_n \vee D'_1 \vee \ldots D'_t)\sigma\}}{N \cup \{(\neg A_{m+1} \vee \ldots \neg A_n)\sigma \vee \neg P_1 \vee \ldots \vee \neg P_t, P_1 \vee D'_1\sigma, \ldots, P_t \vee D'_t\sigma\}}$$

if $P_1, \ldots, P_t$ are new predicate symbols for $D_1\sigma, \ldots, D_t\sigma$, respectively.

The following procedure partitions top-variable clauses.

Algorithm 3: The FindClosedT function

```
Input: A top-variable literal L and subclause C
Output: A closed top-variable subclause C_i
Function FindClosedT(L, C):
    NewTopVar ← Top variables in L
    LinkedTopVarLit ← L
    while NewTopVar ≠ ∅ do
        L_new ← Literals in C that contains NewTopVar
        if L_new ⊆ LinkedTopVarLit then
            NewTopVar ← ∅
        else
            NewTopVar ← Top variables in L_new
            LinkedTopVarLit ← LinkedTopVarLit ∪ L_new
    return LinkedTopVarLit
```

In the **T-Trans** rule a top-variable subclause is partitioned into closed top-variable subclauses. This partition is achieved by traversing all top-variable literals and checking if a top-variable literal belongs a closed top-variable subclause. **Algorithm** 3 gives the FindClosedT(L, C) function, finding L-occurring closed top-variable subclause in the top-variable subclause C.

Algorithm 4: Partitioning a top-variable subclause

Input: A top-variable subclause C
Output: Closed top-variable subclauses $C_1, \ldots, C_n$

$i \leftarrow 1$
while $C \neq \emptyset$ **do**
  Pick a top-variable literal L from C
  $C_i = \text{FindClosedT}(L, C)$
  return C_i
  $C \leftarrow C / C_i$
  $i \leftarrow i + 1$

Algorithm 4 gives the partitioning procedure for a top-variable subclause. The following example shows how **Algorithm** 4, together with the **T-Trans** rule, handles the top-variable resolvents (with respect to a top-variable resolution inference to an indecomposable CO clause as the main premise and guarded clauses as the side premises). Consider an indecomposable CO clause

$$\begin{aligned} Q = {} & \neg A_1(x_1, x_2) \vee \neg A_2(x_1, x_3) \vee \neg A_3(x_2, x_3) \vee \\ & \neg A_4(x_3, x_4) \vee \neg A_5(x_3, x_5) \vee \neg A_6(x_4, x_5) \vee \neg B(x_3) \end{aligned}$$

and the following set N of guarded clauses

$$\begin{aligned} & C_1 = A_1(f(x, y), f(x, y)) \vee B_1(h_1(x, y)) \vee \neg G_1(x, y), \\ & C_2 = A_2(f(x, y), x) \vee \neg G_2(x, y), \qquad C_3 = A_3(f(x, y), x) \vee \neg G_3(x, y), \\ & C_4 = A_4(x, f(x, z)) \vee \neg G_4(x, z), \qquad C_5 = A_5(x, f(x, z)) \vee \neg G_5(x, z), \\ & C_6 = A_6(f(x, z), f(x, z)) \vee B_2(h_2(x, z)) \vee \neg G_6(x, z), \\ & C_7 = B(g(x)) \vee \neg G_7(x). \end{aligned}$$

Figure 4.7 shows the hypergraph that is associated with Q.

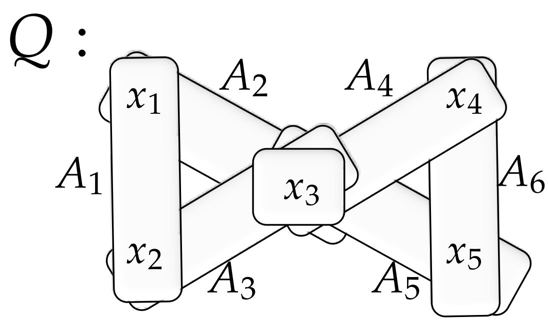

Figure 4.7: The hypergraph associated with Q

Next, we compute the top-variable resolvent of N and Q, and we use **I** to denote this top-variable resolution inference step. By the CompT(N, Q) function, we compute the mgu

$$\{x_1 \mapsto f(g(x), y), x_2 \mapsto f(g(x), y), x_3 \mapsto g(x), x_4 \mapsto f(g(x), z), x_5 \mapsto f(g(x), z)\}$$

for variables in Q. Then in Q, x_1, x_2, x_4 and x_5 are top variables (with respect to **I**) since they are unified with the deepest terms. These top variables occur in all literals except $\neg B(x_3)$, therefore P-Res(N, Q) returns

$$\neg A_1(x_1, x_2), \neg A_2(x_1, x_3), \neg A_3(x_2, x_3), \neg A_4(x_3, x_4), \neg A_5(x_3, x_5), \neg A_6(x_4, x_5)$$

as the top-variable literals (with respect to **I**). Then applying the top-variable resolution inference to Q and N produces

$$\begin{aligned} R = {} & B_1(h_1(x, y)) \vee \neg G_1(x, y) \vee \neg G_2(x, y) \vee \neg G_3(x, y) \vee \\ & B_2(h_2(x, z)) \vee \neg G_6(x, z) \vee \neg G_4(x, z) \vee \neg G_5(x, z) \vee \neg B(x). \end{aligned}$$

Algorithm 4 first finds the closed top-variable subclauses in Q, and then the **T-Trans** rule transforms R into GQ clauses. In this example, the input of **Algorithm** 4 is the top-variable subclause

$$\begin{aligned} Q_{\mathsf{top}} = {} & \neg A_1(x_1, x_2) \vee \neg A_2(x_1, x_3) \vee \neg A_3(x_2, x_3) \vee \\ & \neg A_4(x_3, x_4) \vee \neg A_5(x_3, x_5) \vee \neg A_6(x_4, x_5). \end{aligned}$$

in Q with respect to **I**. In **Algorithm** 4, Line 3 first picks an arbitrary top-variable literal, for example $\neg A_1(x_1, x_2)$, in Q_{top}, and Line 4 then use the FindClosedT function finds a closed top-variable subclause that contains $\neg A_1(x_1, x_2)$. The FindClosedT($\neg A_1(x_1, x_2), Q_{\text{top}}$) function find this closed top-variable subclause as follows. In **Algorithm 3**, Line 2 first identifies that $\neg A_1(x_1, x_2)$ contains top variables x_1 and x_2, and Line 5 then finds literals in Q_{top} that contains x_1 and x_2. Since $\neg A_2(x_1, x_3)$ and $\neg A_3(x_2, x_3)$ contain x_1 and x_2,

$$Q^1_{\text{close}} = \neg A_1(x_1, x_2) \vee \neg A_2(x_1, x_3) \vee \neg A_3(x_2, x_3)$$

is a temporary closed top-variable subclause. Next, **Algorithm 3** keeps looking for the top variables in Q^1_{close}, which are x_1 and x_2. There are no new literals in Q_{top} that contain x_1 or x_2, hence the FindClosedT($\neg A_1(x_1, x_2), Q_{\text{top}}$) function returns Q^1_{close} as the first closed top-variable subclause of Q_{top}. Then Line 6 in **Algorithm 4** removes Q^1_{close} from Q_{top}, obtaining

$$Q'_{\text{top}} = \neg A_4(x_3, x_4) \vee \neg A_5(x_3, x_5) \vee \neg A_6(x_4, x_5).$$

Like the previous procedure, a top-variable literal $\neg A_4(x_3, x_4)$ is picked from $\neg A_4(x_3, x_4) \vee \neg A_5(x_3, x_5) \vee \neg A_6(x_4, x_5)$ and the FindClosedT($\neg A_4(x_3, x_4), Q'_{\text{top}}$) function is applied to find the closed top-variable subclause containing $\neg A_4(x_3, x_4)$. Eventually, the FindClosedT($\neg A_4(x_3, x_4), Q'_{\text{top}}$) function returns

$$Q^2_{\text{close}} = \neg A_4(x_3, x_4) \vee \neg A_5(x_3, x_5) \vee \neg A_6(x_4, x_5).$$

Algorithm 4 splits Q_{top} into closed top-variable subclauses Q^1_{close} and Q^2_{close}.

Using these closed top-variable subclauses Q^1_{close} and Q^2_{close}, the **T-Trans** rule requires us to find the remainders in side premises (of the **I** inference) that match them. The closed top-variable subclause Q^1_{close} contains top-variable literals $\neg A_1(x_1, x_2), \neg A_2(x_1, x_3)$ and $\neg A_3(x_2, x_3)$, which match side premises C_1, C_2 and C_3, respectively. Then for the top-variable resolvent R,, the **T-Trans** rule introduces a fresh predicate symbol P_1 (and a respective literal $\neg P_1(x, y)$) for the disjunction of the remainders of C_1, C_2 and C_3, namely

$$B_1(h_1(x, y)) \vee \neg G_1(x, y) \vee \neg G_2(x, y) \vee \neg G_3(x, y).$$

Similarly, since the literals in Q^2_{close} match to C_4, C_5 and C_6, for the top-variable resolvent R, the **T-Trans** rule introduces a fresh predicate symbol P_2 (and a respective literal $\neg P_2(x,z)$) for the remainders of C_4, C_5 and C_6, namely

$$B_2(h_2(x,z)) \lor \neg G_6(x,z) \lor \neg G_4(x,z) \lor \neg G_5(x,z).$$

Then the **T-Trans** rule transforms R into guarded clauses

$$D_1(h_1(x,y)) \lor \neg G_1(x,y) \lor \neg G_2(x,y) \lor \neg G_3(x,y) \lor P_1(x,y),$$
$$D_2(h_2(x,z)) \lor \neg G_6(x,z) \lor \neg G_4(x,z) \lor \neg G_5(x,z) \lor P_2(x,z),$$

and a query clause $\neg B(x) \lor \neg P_1(x,y) \lor \neg P_2(x,z)$.

By the **T-Trans** rule, the top-variable resolvent of a CO clause and a guarded clausal set is replaced by its equisatisfiable GQ clausal set, formally stated as:

Lemma 4.25. *Let R be the resolvent of applying the **P-Res** rule (endowed with the **T-Ref$^{\mathsf{GQ}}$** refinement) to an indecomposable CO clause Q and a set N of guarded clauses. Then, the following conditions hold.*

1. *Applying the **T-Trans** rule to R replaces it by a set N' of guarded clauses and a query clause Q'.*
2. *Applying the **Q-Sep** procedure to Q' separates it into a set N'' of HG clauses and an indecomposable CO clause Q''.*
3. *The top-resolvent R is satisfiable if and only if the GQ clausal set $N' \cup N'' \cup Q''$ is satisfiable.*
4. *For each clause C' in $N' \cup N''$, there exists a clause C in N such that C' is no wider than C, and Q'' is less wide than Q.*

Proof. Recall the **P-Res** rule (with a priori checking for maximality and the **T-Ref$^{\mathsf{GQ}}$** refinement).

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

if the following conditions are satisfied.

1. No literal is selected in $D_1, \ldots, D_n$ and $B_1, \ldots, B_n$ are strictly $\succ_{lpo}$-maximal with respect to $D_1, \ldots, D_n$, respectively.

2a. If $n = 1$, i) either $\neg A_1$ is selected, or nothing is selected in $\neg A_1 \vee D$ and $\neg A_1\sigma$ is $\succ_{lpo}$-maximal with respect to $D\sigma$, and ii) $\sigma = \mathrm{mgu}(A_1 \doteq B_1)$ or

2b. if $n > 1$ and there exists an mgu σ' such that $\sigma' = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$, then $\sigma = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_m \doteq B_m)$ where $m \leq n$.

3. All premises are variable disjoint.

Suppose in a **P-Res** inference (endowed with the **T-Ref**$^{\mathbf{GQ}}$ refinement), the main premise $C = \neg A_1 \vee \ldots \vee \neg A_m \vee \ldots \vee \neg A_n$ is a CO clause and side premises $C_1 = B_1^* \vee D_1, \ldots, C_m = B_m^* \vee D_m, \ldots, C_n = B_n^* \vee D_n$ are guarded clauses. We use R to denote the top-variable resolvents $(D_1 \vee \ldots \vee D_m \vee \neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma$.

1.: By the **T-Ref**$^{\mathbf{GQ}}$ refinement, a side premise is either a flat ground clause, or a compound-term clause. Suppose a side premise in $C_1 \ldots, C_m$ is flat and ground. Then by 5. in **Lemma 4.13**, all side premises are flat and ground, and R is a flat ground clause, which is an LGQ clause. Now consider the case that each side premise contains at least one compound terms. By **Lemma 4.5**, B_i is a compound-term literal for all i such that $1 \leq i \leq n$.

Step 1: We now prove that $(\neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma$ is a query clause. By 3. in **Lemma 4.13**, the mgu σ substitutes variables in $\neg A_{m+1} \vee \ldots \vee \neg A_n$ with either variables or constants. Hence, $(\neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma$ is a query clause. If σ is a ground substitution, then $(\neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma$ is a flat ground clause. If all literals in C are top-variable literal, then $(\neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma$ is $\perp$. Our statement trivially holds for these two special cases.

Step 2: we prove that $(D_1 \vee \ldots \vee D_m)\sigma$ is a disjunction of guarded clauses. We prove this by the following steps: 2-i) $D_i\sigma$ is a guarded clause, 2-ii) $(D_i \vee D_j)\sigma$ a guarded clause if A_i and A_j contains connected top variables, and 2-iii) Suppose $\neg A_{i_1} \vee \ldots \vee \neg A_{i_k}$ is a closed top-variable subclause of $\neg A_1 \vee \ldots \vee \neg A_m$, and D'_j represents $D_{i_1} \vee \ldots \vee D_{i_k}$. Then $(D_1 \vee \ldots \vee D_m)\sigma$ can be represented as $(D'_1 \vee \ldots \vee D'_t)\sigma$ where $t \leq m$.

Step 2-i): First we prove that remainder D_i of side premise is a guarded clause. By the covering property and the fact the B_i is a compound-term literal, $\mathrm{var}(B_i) = \mathrm{var}(C_i)$. By 4. in **Lemma 4.13**, the mgu σ substitutes variables in C_i with variables and constants. Then by the fact that C_i is a guarded clause and **Lemma 4.12**, $D_i\sigma$ is a guarded clause.

Step 2-ii): Next we prove that if A_i and A_j contains connected top variables, the disjunction of $D_i\sigma$ and $D_j\sigma$ is a guarded clause. Suppose x and y are, respectively, top variables in A_i and A_j, and x and y are connected. By the definition of connected top variables, there exists a sequence of top variables $x, \ldots, y$ such that each pair of adjacent variables co-occurs in a top-variable literal. By *3.* in **Lemma 4.13**, each adjacent variables x' and y' have the property that $\text{var}(x'\sigma) = \text{var}(y'\sigma)$, since x' and y' match compound terms in the same covering literal. Hence, $\text{var}(x\sigma) = \text{var}(y\sigma)$. This implies that after unification, the compound term t in B_i (that x matches) have the same variable set as the compound term s in B_j (that y matches), i.e., $\text{var}(s\sigma) = \text{var}(t\sigma)$. By the covering property, $\text{var}(D_i\sigma) = \text{var}(D_j\sigma)$. The guarded clauses $D_i\sigma$ and $D_j\sigma$ contain the same variable sets, therefore, $D_i\sigma \vee D_j\sigma$ is a guarded clause.

Step 2-iii): Suppose $\neg A_{i_1} \vee \ldots \vee \neg A_{i_k}$ is a closed top-variable subclause of $\neg A_1 \vee \ldots \vee \neg A_m$, and D'_j represents $D_{i_1} \vee \ldots \vee D_{i_k}$. We aim to prove that $(D_1 \vee \ldots \vee D_m)\sigma$ can be represented as $(D'_1 \vee \ldots \vee D'_t)\sigma$ where $t \leq m$. We use C' to denote a top-variable subclause $\neg A_1 \vee \ldots \vee \neg A_m$. By the fact that each literal in C' contains at least one top variable, and 2b. of **Definition 17** that each pair of closed top-variable subclauses of C' shares no connected top variables, C' is partitioned into a set of closed top-variable subclauses, denoted as $C'_1, \ldots, C'_t$. Let $C'_i = \neg A_{i_1} \vee \ldots \vee \neg A_{i_k}$ be a closed top-variable subclause of C'. By 2a. of **Definition 17**, each pair of top variables in C'_i is connected. Then by the result of Step 2-ii), $(D_{i_1} \vee \ldots \vee D_{i_k})\sigma$ is a guarded clause.

Now we present the resolvent as $R = (D'_1 \vee \ldots \vee D'_t \vee \neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma$. Then applying the **T-Trans** rule to R transforms it into

$$D'_1\sigma \vee P_1,\ \ldots,\ D'_t\sigma \vee P_t,\ Q' = (\neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma \vee \neg P_1 \vee \ldots \vee \neg P_t.$$

We consider whether $D'_i\sigma \vee P_i$ and Q' are LGQ clauses. Suppose $D'_i\sigma$ is ground. Then immediately $D'_i\sigma \vee P_i$ is a guarded clause. Now assume that $D'_i\sigma \vee P_i$ is non-ground. By the result of Step 2-iii), $D'_i\sigma$ is a guarded clause. By the definition of structural transformation, $\text{var}(D'_i\sigma) = \text{var}(P_i)$ and P_i is a flat literal, hence, $D'_i\sigma \vee P_i$ is a guarded clause. Next we consider Q'. By the definition of structural transformation, $\neg P_1 \vee \ldots \vee \neg P_t$ is a negative flat clause. Then by the result of Step 1, Q' is a query clause.

2.: By **Lemma 4.23**.

3.: By the facts that the **Q-Sep** procedure (**Lemma 4.19**) and the structural transformation preserve logical equivalence.

4.: First we prove that $D'_i\sigma \vee P_i$ is no wider than one of side premises in $C_1, \ldots, C_m$. By the result of Step 2-i), i) the loose guard $\mathbb{G}\sigma$ in $D'_i\sigma$ is inherited from one of loose guards $\mathbb{G}$ in a side premise in $C_1, \ldots, C_m$, and ii) the mgu substitutes variables in $\mathbb{G}$ with either constants or variables. Hence, there exists at least one side premise in $C_1, \ldots, C_m$ such that it contains no less types of variables than $D'_i\sigma \vee P_i$.

Next, we consider the width of the conclusions of applying the **Q-Sep** procedure to $Q' = (\neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma \vee \neg P_1 \vee \ldots \vee \neg P_t$. To understand the application of the **Q-Sep** procedure to Q', we analyse the variables in Q'. By the **T-Trans** rule, $\text{var}(P_i) = \text{var}(D'_i\sigma)$ for all i such that $1 \leq i \leq t$. Hence, w.l.o.g. we consider variables in P_1 (i.e., variables in $D'_1\sigma$), and we suppose $D'_1 = D_1 \vee D_2$ and $\neg A_1 \vee \neg A_2$ is a closed top variable subclause. Hence, we analyse overlapping variables of $D'_1\sigma$ and $(\neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma$. By *4.* of **Lemma 4.13**, σ substitutes variables in D'_1 with either variables or constants. W.l.o.g. suppose variables of B_1 and B_2 are substituted with the variables of $\neg A_1 \vee \neg A_2$. Since D_1 and D_2, respectively, contain loose guards of B_1 and B_2, σ substitutes the variables of D'_1 with the variables of $\neg A_1 \vee \neg A_2$. Then the variables in $D'_1\sigma$ depend on two factors: 1) whether all variables in B_1 and B_2 are substituted, 2) whether (non-top) variables in $\neg A_1 \vee \neg A_2$ occur in the variables of $\neg A_{m+1} \vee \ldots \vee \neg A_n$. We distinguish two cases:

4-i): Suppose all variables in B_1 and B_2 are substituted using σ, and all non-top-variables in $\neg A_1 \vee \neg A_2$ occur in $\neg A_{m+1} \vee \ldots \vee \neg A_n$. Then all variables in $D'_1\sigma$ are non-top-variables in $\neg A_1 \vee \neg A_2$, therefore, $\text{var}(P_1) \subseteq \text{var}((\neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma)$. For each P_i in $P_1, \ldots, P_t$ that satisfies conditions of 4-i), $\text{var}(P_i) \subseteq \text{var}((\neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma)$. These P_i literals introduce no new variables to $\neg A_{m+1} \vee \ldots \vee \neg A_n$.

4-ii): Next suppose not all non-top-variables in $\neg A_1 \vee \neg A_2$ occur in $\neg A_{m+1} \vee \ldots \vee \neg A_n$, or only a part of variables in B_1 and B_2 are substituted. Then $\neg P_1$ can be represented as $\neg P_1(\overline{x}, \overline{y})$, where $\overline{x}$ represent substituted variables in B_1 and B_2 that occur in both $\neg A_1 \vee \neg A_2$ and $(\neg A_{m+1} \vee \ldots \vee \neg A_n)$, and $\overline{y}$ represents either variables that are not substituted in B_1 and B_2, or variables in $\neg A_1 \vee \neg A_2$ but not in $\neg A_{m+1} \vee \ldots \vee \neg A_n$. Hence, $\overline{y}$ does not occur in $(\neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma \vee \neg P_2 \vee \ldots \vee \neg P_t$. Applying the **QuerySepTwo** rule to Q' derives Horn guarded clauses $\neg P_1(\overline{x}, \overline{y}) \vee P'_1(\overline{x})$ and a query clause $(\neg A_{m+1} \vee \ldots \vee \neg A_n)\sigma \vee \neg P'_1(\overline{x}) \vee \ldots \vee \neg P_t$ (using a new predicate symbol P'_1). By Step 2-iii) and the fact that $\neg P_1(\overline{x}, \overline{y}) \vee P_1'(\overline{x})$ is no wider than $D_1'\sigma \vee P_1$, there exists at least one side premises $C_1, \ldots, C_m$ that is wider than $\neg P_1(\overline{x}, \overline{y}) \vee P_1'(\overline{x})$. After separating all P_i satisfying 4-ii), we obtained query clause Q''. Since Q'' contains only non-top-variables (after substituted by σ) from Q, Q'' is less wider than Q. □

An alternative approach for the **T-Trans** rule is the **Sep** rule. Let us use

$$R = (\neg A_{m+1} \vee \ldots \vee \neg A_n \vee D_1' \vee \ldots D_t')\sigma$$

in the definition of the **T-Trans** rule to denote the top-variable resolvent of an indecomposable CO clause and a set of guarded clauses. By the proof in **Lemma** 4.25, recursively applying the **Sep** rule to R also separates R into GQ clauses. In this thesis, we choose the **T-Trans** rule for its intuitiveness.

We use **Q-CO$^{\mathsf{GQ}}$** to denote the following procedure:

1. Apply the top-variable resolution rule to an indecomposable CO clause (as the main premise) and guarded clauses (as the side premises), deriving the top-variable resolvent R.

2. Apply the **T-Trans** rule to R, replacing R by a query clause Q and a set of guarded clauses.

3. Apply the **Q-Sep** procedure to Q, replacing Q by HG clauses and an indecomposable CO clause.

Lemma 4.26. *The conclusions of applying the* **Q-CO$^{\mathsf{GQ}}$** *procedure to an indecomposable CO clause Q and a set N of guarded clause satisfy the following conditions.*

1. *They consist of an indecomposable CO clause Q' and a set N' of guarded clauses.*
2. *The clausal sets $Q' \cup N'$ and $Q \cup N$ are equisatisfiable.*
3. *For each clause C' in N', there exists a clause C in N such that C' is no wider than C, and Q' is less wide than Q.*

Proof. By **Lemmas** 4.23 and 4.25, *1.* and *3.* hold. By **Lemma** 3.4 and the fact that any form of structural transformation rule preserves satisfiability, *2.* hold. □

4.6 A decision procedure of answering BCQs for GF

Now we are ready to give our saturation-based procedure for answering a union of BCQs of GF, and we use **Q-Ans$^{\mathsf{GF}}$** to denote this procedure. To show that the **Q-Ans**$^{\mathsf{GF}}$ procedure is suitable for implementations in modern saturation-based first-order provers, this procedure is devised in the form of the given-clause algorithm [Wei01, MW97].

Algorithm 5 describes the formal BCQ answering procedure for GF.

Algorithm 5: The BCQ answering procedure for GF

Input: A union q of BCQs and a set Σ of formulas in GF
Output: 'Yes' or 'No'

```
workedOff ← ∅
usable ← PreProcessGF(Σ, q)
while usable = ∅ and ⊥ ∉ usable do
    given ← Pick(usable)
    workedOff ← workedOff ∪ given
    if given is an indecomposable CO clause then
        tResolvent ← P-Res(workedOff, given)
        G, Q ← T-Trans(tResolvent)
        CO, HG ← Sep(Q)
        new ← G ∪ CO ∪ HG
    else
        new ← P-Res(workedOff, given) ∪ Fact(given)
    new ← Red(new, new)
    new ← Red(Red(new, workedOff), usable)
    workedOff ← Red(workedOff, new)
    usable ← Red(usable, new) ∪ new
Print(usable)
```

Functions in **Algorithm** 5 are described as follows.

1. Sep(C) applies the **Q-Sep** procedure to separate a query clause C, and returns HG clauses and indecomposable CO clauses. If C is an indecomposable CO clause or an HG clause, then the Sep(C) function returns C.

2. Pick(N) picks a clause from the clausal set N, and then removes that clause from N.

3. P-Res(N, C) applies the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{GQ}}$ refinement) to clauses N and a clause C in N, returning the top-variable resolvent.

4. T-Trans(C) applies the **T-Trans** rule to the top-variable resolvent C, and returns guarded clauses and a query clause.

5. Fact(C) applies the **Fact** rule (endowed with the **T-Ref**$^{\mathsf{GQ}}$ refinement) to a clause C, and returns a factor of C.

6. Red(N_1, N_2) returns all clauses from N_1 that are not redundant with respect to clauses in N_2.

7. Print(N) takes a saturated clausal set N as input, and returns either 'Yes' or 'No' for the BCQ answering problem.

We now give the PreProcessGF function. See **Algorithm 6**.

Algorithm 6: The PreProcessGF function

Input: A union q of BCQs and a set Σ of guarded formulas
Output: A set of indecomposable CO clauses and guarded clauses

Function PreProcessGF(Σ, q)**:**
  usable $\leftarrow \emptyset$
  G, Q $\leftarrow$ TransGF(Σ, q)
  foreach *clause* Q *in* Q **do**
    CO, HG $\leftarrow$ Sep(Q)
    usable $\leftarrow$ usable $\cup$ CO $\cup$ HG
  usable $\leftarrow$ Red(usable $\cup$ G, usable $\cup$ G)
  return usable

The PreProcessGF(Σ, q) function pre-processes the given set Σ of guarded formulas and the given union q of BCQs to a set of guarded clauses and indecomposable CO clauses. In the PreProcessGF(Σ, q) function, the TransGF(Σ, q) function applies the **Trans**$^{\mathsf{GF}}$ process to Σ and q, returning GQ clauses.

Algorithms 5–6 use the notations G, HG, Q and CO to denote guarded, Horn guarded, query and indecomposable chained-only query clauses, respectively.

Algorithm 7 gives the Print function, simply prints 'Yes' or 'No' for the BCQ answering problem.

Algorithm 7: The Print function

Input: A clausal set N
Output: 'Yes' or 'No'
Function Print(N)**:**
  if $\bot \in N$ **then** Print 'Yes'
  else Print 'No'

As a given-clause algorithm, **Algorithm 5** splits input clauses into a worked-off clause set *workedOff* where all inferences between clauses in *workedOff* are finished, and a usable clause set *usable*, in which clauses needed to be considered for further inferences. Then for each clause C in *usable*, we remove C from *usable*, add C to *workedOff* and then add all conclusions between the C and the clauses in *workedOff* to *usable*. During such a loop, reduction rules are applied to guarantee termination.

Algorithm 5 consists of the following stages.

- Lines 1–3 transform BCQs and GF into CO clauses and guarded clauses.
- Lines 4–17 saturate the above set of CO clauses and guarded clauses.
- Line 18 outputs the answer to the given BCQs and guarded formulas.

Lines 1 initialises the *workedOff* and *usable* sets with empty sets. Line 2, namely **Algorithm 6**, then transforms BCQs and GF to CO clauses and guarded clauses, and then add all the obtained clauses to the *usable* set. Line 3 is the *input reduction* that removes redundancy in the *usable* set.

The while-loop in Lines 4–13 terminates if either *usable* is empty or it contains an empty clause $\bot$. Lines 5–6 pick a clause, namely the *given* clause, from the *usable* set and then add the *given* clause to *workdedOff*. Note that the Pick function is *fair* [BG01, Page 37], which means that no clause in the *usable* set waits infinitely long without being picked. Lines 7–13 generate new conclusions. Lines 7–11 say that if *given* is an indecomposable CO clause, then the **Q-CO**$^{\mathsf{GQ}}$ procedure, namely a sequence of rules consisting of i) the top-variable resolution inference, ii) the **T-Trans** rule and iii) the **Q-Sep** procedure, is applied to this CO clause, producing an indecomposable CO clause and guarded clauses, and we then denote these new clauses as *new*. Lines 12–13 say that if *given* is a guarded clause, then **P-Res** and **Fact** rules are applied to that guarded clause, producing new conclusions *new*. Lines 14–17 are the *inter-reduction step* that removes redundancy in the *new*, *workdedOff* and *usable* clausal sets.

Lines 18, namely **Algorithm 7**, outputs the answer to the given BCQs. Suppose q is the given union of BCQs and Σ is the given guarded formulas. An empty *usable* clausal set implies that $\Sigma \cup \{\neg q\}$ is satisfiable. Then the answer to q is 'No', and the *workdedOff* clausal set stores the saturated clausal set of $\Sigma \cup \{\neg q\}$. Otherwise the *usable* clausal set contains an empty clause, then $\Sigma \cup \{\neg q\}$ is unsatisfiable. Then the answer to q is 'Yes'.

The **Q-Ans**$^{\textsf{GF}}$ procedure guarantees termination. The termination result requires that given a union of BCQs, a set of guarded formulas and a finite signature $(\mathtt{F}, \mathtt{P}, \mathtt{C})$, the **Q-Ans**$^{\textsf{GF}}$ procedure derives clauses of bounded depth and width only using symbols in $(\mathtt{F}, \mathtt{P}, \mathtt{C})$. In **Algorithm 5**, Lines 7–13 derive new clauses. In Lines 14–17, given an indecomposable CO clause and guarded clauses, the **Q-CO**$^{\textsf{GQ}}$ procedure produces GQ of bounded width, as proved in **Lemma 4.26**. Since the class of GQ clauses are simple, their depths are bounded as well. For Lines 12–13, **Theorem 4.4** ensures that given guarded clauses, the **P-Res** and **Fact** rules (endowed with the **T-Ref**$^{\textsf{GQ}}$ refinement) produce guarded clauses of bounded width and depth. To sum up, Lines 7–13 derive new GQ clauses of bounded width and depth. Next it is important to ensure that the number of new symbols introduced in **Q-Ans**$^{\textsf{GF}}$ procedure are finitely many. In particular, we consider the new predicate symbols that are introduced in the derivation, particularly for the **T-Trans** rule in Line 9 and for the **Q-Sep** procedure in Line 10.

In the **Q-Ans**$^{\textsf{GF}}$ procedure, suppose a predicate symbol P is used to represent a GQ clause C at a derivation stage. Then, in any further stage whenever a predicate symbol is needed for C, the symbol P is used again. The **Q-Ans**$^{\textsf{GF}}$ procedure requires only a finite number of predicate symbols, formally stated as:

Lemma 4.27. *In the application of the **Q-Ans**$^{\textsf{GF}}$ procedure to the BCQ answering problem for GF, only finitely many predicate symbols are introduced.*

Proof. In the **Q-Ans**$^{\textsf{GF}}$ procedure, new predicate symbols are introduced in either the **Trans**$^{\textsf{GF}}$ process, or the **Q-Sep** procedure, or the **Q-CO**$^{\textsf{GQ}}$ procedure. We distinguish these cases:

1.: In the **Trans$^{\mathsf{GF}}$** process, BCQs and formulas in GF are transformed into GQ clauses. This is a one-time process, hence in this step, only a finite number of new predicate symbols are introduced.

2.: In the **Q-Sep** procedure, the **QuerySepOne** and **QuerySepTwo** rules are recursively applied to query clauses. By **Lemma** 4.23, applying the **Q-Sep** procedure to a query clause derives less wide GQ clauses. Then finitely many new predicate symbols are needed.

3.: In the **Q-CO$^{\mathsf{GQ}}$** procedure, the **T-Trans** rule and the **Q-Sep** procedure are applied. By **Lemma** 4.23, the **Q-Sep** procedure introduces finitely many new predicate symbols. Now we consider the new predicate symbols introduced in the application of the **T-Trans** rule to the top-variable resolvents of a CO clause and guarded clauses. Consider the **T-Trans** rule as follows: Let N be a set of compound-term guarded clauses, as the side premises in the top-variable resolution rule. Then in this top-variable inference, we first unify clauses in N, and then removes a compound-term literal (eligible literals) in each clause of N, respectively, and finally the **T-Trans** rule makes remainder of clauses (that map to the same closed top-variable subclause) in N a disjunction. W.l.o.g. assume the **T-Trans** rule introduce a new predicate symbol for C_1 and C_2 in N, producing C. By *1.* of **Lemma** 4.25, C is no wider than C_1 and C_2. Hence we can regard the **T-Trans** rule as a method that i) first unifies a compound-term guarded clausal set N by their compound-term literals, ii) remove these unified compound-term literal in clauses of N, obtaining N', ii) use the disjunctive symbol to connect clauses in N', as a new guarded clause C. By the covering and pairing properties, no new variables are needed in i)–iii). By **Lemma** 4.23, **Lemma** 4.26 and **Theorem** 6.4, apart from the **T-Trans** rule, the **Q-Ans$^{\mathsf{GF}}$** procedure guarantees producing a finite number of GQ clauses. As we reuse predicate symbols for duplicate GQ clauses, the **T-Trans** rule introduces a finite number of predicate symbols. □

Finally, we give the main result of this chapter.

Theorem 4.5. *The* ***Q-Ans$^{\mathsf{GF}}$*** *procedure is a decision procedure for answering BCQs for GF.*

Proof. By **Theorem** 4.1, the **Q-Ans$^{\mathsf{GF}}$** procedure reduces the problem of answering BCQs for GF to that of deciding satisfiability of the GQ clausal class.

By **Lemma** 4.19 and **Theorem** 4.3, the **T-Inf**$^{\mathsf{GQ}}$ system is a sound and refutationally complete system for general first-order clausal logic. As the **Q-Ans**$^{\mathsf{GF}}$ procedure is based on the **T-Inf**$^{\mathsf{GQ}}$ system and our customised separation rules, the **Q-Ans**$^{\mathsf{GQ}}$ procedure is a sound and refutational complete procedure if only finitely many predicate symbols are introduced.

By **Lemma** 4.23, **Lemma** 4.26 and **Theorem** 4.4, applying the **Q-Ans**$^{\mathsf{GF}}$ procedure to GQ clauses guarantees producing GQ clauses of bounded depth and bounded width. By **Lemma** 4.27, only a finitely number of new symbols (with respect to the given signature) are introduced. Then the **Q-Ans**$^{\mathsf{GF}}$ procedure guarantees termination. Since the **Q-Ans**$^{\mathsf{GF}}$ procedure is sound, refutationally complete for first-order clausal logic and guarantees termination for the GQ clausal class, it is a decision procedure for answering BCQs for GF. □

Chapter 5

The saturation-based BCQ rewriting procedure in GF

In this chapter, we aim to address the saturation-based rewriting problem for GF, formally defined as follows.

Problem 5. *Given a set Σ of formulas in GF, a set* D *of ground atoms and a union q of BCQs, does there exist a (function-free) first-order formula (with equality) Σ_q, which is the negated back-translation of the saturated clausal set of $\Sigma \cup \{\neg q\}$ such that $\Sigma \cup \mathrm{D} \models q$ if and only if $\mathrm{D} \models \Sigma_q$?*

Problem 5 considers the back-translation of a clausal set to a function-free first-order formula. Consider formulas Σ in GF, ground atoms D and a union q of BCQs, we use the following steps to tackle **Problem 5**.

1. Apply the **Q-Ans**$^{\mathsf{GF}}$ procedure to $\Sigma \cup \{\neg q\}$, computing a saturation N of $\Sigma \cup \{\neg q\}$ as long as $\Sigma \not\models q$.

2. Back-translate N to a (Skolem-symbol-free) first-order formula F, and then negate F to obtain a (function-free) first-order formula (with equality) Σ_q.

As guarded formulas and BCQs contain no function symbol, the function symbols in N are introduced by Skolemisation. Back-translating N to a first-order formula F ensures to eliminate all Skolem function symbols in F, therefore Σ_q a function-free first-order formula. In general, the back-translation from a clausal set to a first-order formula is a non-trivial task, as it often fails [GSS08a]. By **Theorem 3.1**, our aim is to transform the saturation N to its logically equivalent clausal set that is unique, normal, globally linear and globally compatible, so that N can be back-translated to a first-order formula.

In this chapter, we investigate a more refined clausal form for guarded clauses, that is the *aligned guarded clauses*, in which all compound terms have a common argument list. We prove that the **Q-Ans**$^{\mathsf{GF}}$ procedure is also a decision procedure for the *aligned guarded clausal class*. Then, by our customised **Rename**, **Abstract** and **Unsko** rules (from **Section** 3.3), any aligned guarded clausal set N is ensured to transformed into a unique, normal, globally linear and globally compatible clausal set N' that is logically equivalent to N. In the last step, N' is unskolemised to a first-order formula.

We use the notation N_{a} to denote a clausal set N that has the property a. In particular we use n, u, l and c to denote the properties of normality, uniqueness, global linearity and global compatibility, respectively. For example, N_{nu} denotes a clausal set N that is unique and normal.

This chapter is organised as follows. **Section** 5.1 presents the formal definition of the aligned guarded clauses, and **Section** 5.2 then formally proves that the **Q-Ans**$^{\mathsf{GF}}$ procedure decides satisfiability of the aligned guarded clausal class. **Section** 5.3 introduces our variations of **Rename**, **Abstract** and **Unsko** rules and the back-translation procedure. The last section formally formalises decision procedure for the saturation-based BCQ rewriting in GF.

5.1 The aligned guarded clauses

In this section, we first introduce the aligned guarded clauses, and then show that by the **Trans**$^{\mathsf{GF}}$ process, guarded formulas are clausified to aligned guarded clauses.

Recall **Definition** 12 that a clause is strongly compatible if all of its compound terms share a common argument list. Then an *aligned guarded clause* is formally defined as follows.

Definition 18. *An* aligned guarded clause (G^- clause) *is strongly compatible and a guarded clause.*

Comparing **Definitions** 13 and 18, the class of G^- clauses is a strict subset of that of guard clauses. This means that the results established in **Section** 4.4 hold for the G^- clausal class as well. The notion of *strongly compatible* is more restrictive than that of *compatible*, since all compound terms must be compatible. For example, the clause

$$C = \neg G(x_1, x_2) \vee A_1(f(x_1, x_1, x_2), x_1) \vee A_2(f(x_1, x_1, x_2), x_1, g(x_1, x_2))$$

is compatible and a guarded clause, however it is not a G^- clause since $f(x_1, x_1, x_2)$ and $g(x_1, x_2)$ are not compatible. The guarded clause $C = \neg G(x, y) \vee A(f(x, y), f(y, x))$ is not a G^- clause, since in C the compound terms $f(x, y)$ and $f(y, x)$ are not compatible. Note that even by the **Rename**, **Abstract** and **Unsko** rules in **Section 3.3**, C cannot be back-translated into a first-order formula as it is impossible to make the argument lists of $f(x, y)$ and $f(y, x)$ identical.

Lemma 5.1. *Applying the* ***Trans***$^{\mathsf{GF}}$ *process to a guarded formula transforms it into a set of* G^- *clauses.*

Proof. We particularly show that the obtained clauses are strongly compatible. The fact that the obtained clauses are guarded clauses follows from **Lemma 4.1**.

Suppose F is a guarded formula. By 1.–2. of the **Trans**$^{\mathsf{GF}}$ process, w.l.o.g. suppose P is the new predicate symbol, F_1 is the definition formula of P, and F' is the replacing formula of F. Next, we show that 3.–4. of the **Trans**$^{\mathsf{GF}}$ process transform F_1 and F' into strongly compatible clauses. Since F' is an existentially quantified sentence, skolemising F' transforms it into flat ground clauses, which are aligned guarded clauses. F_1 can be represented as

$$\forall \overline{x}(P(\overline{x}) \rightarrow \forall \overline{y}(G(\overline{x}, \overline{y}) \rightarrow \phi(\overline{y})))$$

where $\phi(\overline{y})$ may contain existential quantifications and literals, but no universal quantifications. The existential quantified variables in $\phi(\overline{y})$ are the only source of compound terms, since guarded formulas contain no function symbols. By applying prenex normal form transformation and then the **Skolem** rule to F_1, existential quantified variables in $\phi(\overline{y})$ are skolemised to compound terms containing a common argument list $\overline{x}, \overline{y}$. □

We use the notation GQ^- to denote the class of G^- and query clauses.

Theorem 5.1. *The* ***Trans***$^{\mathsf{GF}}$ *process reduces the problem of* BCQ *answering for* GF *to that of deciding satisfiability of the* GQ^- *clausal class.*

Proof. By **Lemma 5.1**. □

5.2 Deciding the GQ$^-$ clausal class

In this section, we aim to prove that the **Q-Ans$^{\mathsf{GF}}$** procedure decides satisfiability of the GQ$^-$ clausal class. By **Theorem 4.5** and the fact that the G$^-$ clausal class is a subset of the guarded clausal class, applying the **Q-Ans$^{\mathsf{GF}}$** procedure to GQ$^-$ clauses derives GQ clauses. Hence in this section, we put our focus on proving that the derived GQ clauses are strongly compatible.

Recall that flat compound terms are non-nested compound terms. We first investigate the unification of flat and compatible compound terms.

Lemma 5.2. *Let s, s', t and t' be flat compound terms. Suppose s and s' are compatible, and t and t' are compatible. Then, if s and t are unifiable by an mgu σ, the following conditions are satisfied.*

1. *s and t are compatible, and $s\sigma$ and $t\sigma$ are compatible.*
2. *$s\sigma$ and $s'\sigma$ are compatible, and $t\sigma$ and $t'\sigma$ are compatible.*
3. *$s'\sigma$ and $t'\sigma$ are compatible.*

Proof. Since s and t are, respectively, compatible with s' and t', $s\sigma$ and $t\sigma$ are compatible with $s'\sigma$ and $t'\sigma$, respectively. By the fact that s and t are unifiable by σ and **Definition 8**, $s\sigma$ and $t\sigma$ are compatible. Then $s'\sigma$ and $t'\sigma$ are compatible. □

Now we look at the conclusions that are obtained by applying the **Q-Ans$^{\mathsf{GF}}$** procedure to GQ$^-$ clauses. In particular we focus on the checking the strong compatibility property of the derived clauses.

By **Lemma 4.6** and the fact that the GQ$^-$ clausal class belong to the guarded clausal class, *a priori checking* is applied in performing the **Fact** and **P-Res** rules to GQ$^-$ clauses. We start with checking whether the **T-Inf$^{\mathsf{GQ}}$** system decides satisfiability of the aligned guarded clauses.

We start with considering the applications of the **Fact** rule to GQ$^-$ clauses.

Lemma 5.3. *Applying the **Fact** rule (endowed with the **T-Ref$^{\mathsf{GQ}}$** refinement) to GQ$^-$ clauses derives GQ$^-$ clauses.*

Proof. Recall the **Fact** rule (with an a priori checking for maximality and the **T-Ref$^{\mathsf{GQ}}$** refinement).

$$\textbf{Fact}: \quad \frac{C \vee A_1 \vee A_2}{(C \vee A_1)\sigma}$$

if the following conditions are satisfied.

1. Nothing is selected in $C \vee A_1 \vee A_2$.
2. A_1 is $\succ_{lpo}$-maximal with respect to C.
3. $\sigma = \text{mgu}(A_1 \doteq A_2)$.

Since the **Fact** rule is not applicable to query clauses, we consider the case when a G^- clause is the premise of the **Fact** rule. By **Lemma 4.15**, the factors of applying the **Fact** rule (endowed with the **T-Ref**$^{\mathsf{GQ}}$ refinement) to G^- clauses are guarded clauses. We prove that these factor are strongly compatible. By **Algorithm 1**, the premise $C \vee A_1 \vee A_2$ is a G^- clause. By **Algorithm 1**, we distinguish two cases of $C \vee A_1 \vee A_2$.

Lines 1–2: The case is trivial when $C \vee A_1^* \vee A_2$ is ground.

Lines 5–6: The premise $C \vee A_1^* \vee A_2$ is non-ground and contains positively occurring compound-term literals, but no negatively occurring compound-term literals. By **Definition 18**, all compound terms in $C \vee A_1 \vee A_2$ share a common argument list, therefore all compound terms in $(C \vee A_1 \vee A_2)\sigma$ are compatible. Hence $(C \vee A_1)\sigma$ is strongly compatible. □

Next we consider the applications the **P-Res** rule to GQ^- clauses. We first check the case when all the premises are G^- clauses, and then consider the case when the main premise is a query clause and the side premises are G^- clauses.

Lemma 5.4. *Applying the* **P-Res** *rule (endowed with the* **T-Ref**$^{\mathsf{GQ}}$ *refinement) to* G^- *clauses derives* G^- *clauses.*

Proof. By **Lemma 4.16**, applying the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{GQ}}$ refinement) to G^- clauses derives guarded clauses. Now we focus on proving that these derived guarded clauses are strongly compatible.

By **Algorithm 1**, the binary form of the **P-Res** rule is used when the premises are G^- clauses. Suppose G^- clauses $C_1 = B_1 \vee D_1$ and $C = \neg A_1 \vee D$ are the positive and the negative premise in the **P-Res** rule, respectively, deriving the resolvent $C' = (D_1 \vee D)\sigma$, where σ is the mgu of B_1 and A_1. By **Algorithm 1**, C either is ground, or has at least one a negatively occurring non-ground compound-term literal or is flat (Lines 1–2, or 3–4 or 7–8, respectively) and C_1 satisfies either Lines 1–2 or 5–6. We distinguish three cases of C:

Lines 1–2: The negative premise C is a ground clause. By **Algorithm 1**, the positive premise C_1 is either a ground flat clause or a ground compound-term clause. The case is trivial when C is flat and ground. When C is a ground compound-term and , this case is a special case when C satisfies Lines 3–4, which is proved next.

Lines 3–4: The negative premise C contains at least one negative non-ground compound-term literal. By **Algorithm 1**, the positive premise C_1 is either i) a simple ground clause, or ii) contains positive non-ground compound-terms, but no negative non-ground compound-terms. In i) C_1 is a ground compound-term clause, otherwise A_1 and B_1 are not unifiable. The case C_1 is a ground compound-term clause is a special case of ii). Next we consider ii). By **Lemma 4.5**, B_1 is a compound-term literal. By **Lemma 4.8**, compound terms in A_1 pair only compound term in B_1, and vice-versa. W.l.o.g. suppose s, s', t and t' are compound terms in B_1, D_1, A_1 and D, respectively. Further suppose s pairs t. By **Definition 18**, s and s' are compatible, and t and t' are compatible. By the fact that s pairs t, $s\sigma$ and $t\sigma$ are compatible. By **Lemma 5.2**, $s'\sigma$ and $t'\sigma$ are compatible. Hence all compound terms in $(D_1 \vee D_2)\sigma$ are compatible. By **Lemma 4.16**, $(D_1 \vee D_2)\sigma$ is an aligned guarded clause.

Lines 7–8: The negative premise C is a flat guarded clause. **Algorithm 1**, C_1 is either i) a simple and ground clause, or ii) contains positive non-ground compound-terms, but no negative non-ground compound-terms. By the fact that C is flat and C_1 is strongly compatible, the compound terms in the resolvent C' are from C_1, therefore C' is strongly compatible. Next suppose C_1 is an aligned guarded clause containing positive non-ground compound-terms, but no negative non-ground compound-terms. Since C is a flat guarded clause, all compound terms in the resolvent C' originate from compound terms in C_1. Suppose s and t are two arbitrary compound terms in C_1. By *2a.* of **Definition 18**, s and t are compatible. Hence $s\sigma$ and $t\sigma$ are compatible. Then all compound terms in C' are compatible. By **Lemma 4.16**, C' is an aligned guarded clause. □

We now conclude the result of applying the **T-Inf**$^{\mathsf{GQ}}$ system to G^- clauses. This is formally stated as:

Theorem 5.2. *The **T-Inf**$^{\mathsf{GQ}}$ system decides satisfiability of the G^- clausal class.*

Proof. By **Theorem** 4.4 and **Lemmas** 5.3–5.4. □

Next we consider the application of the **P-Res** rule to a query clauses and G^- clauses. In the **Q-Ans**$^{\mathsf{GF}}$ procedure, the top-variable resolution rule is performed on an indecomposable CO query clause and guarded clauses, deriving the top-variable resolvent R, and then by the **T-Trans** rule, R is transformed into a set N of guarded clauses (and query clauses). We check that whether the guarded clauses in N are strongly compatible.

Lemma 5.5. *Let R be the top-variable resolvent of applying the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{GQ}}$ refinement) to an indecomposable CO clause and G^- clauses. Then, by the **T-Trans** rule R is replaced by G^- clauses and a query clause.*

Proof. By **Lemma** 4.25 and the fact that the G^- clausal class is a strict subset of the guarded clausal class, R is replaced by a set N of guarded clauses and a query clause. Hence we focuses on proving that clauses in N are strongly compatible.

Recall the **P-Res** rule (with a priori checking for maximality and the **T-Ref**$^{\mathsf{GQ}}$ refinement).

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

if the following conditions are satisfied.

1. No literal is selected in $D_1, \ldots, D_n$ and $B_1, \ldots, B_n$ are strictly $\succ_{lpo}$-maximal with respect to $D_1, \ldots, D_n$, respectively.

2a. If $n = 1$, i) either $\neg A_1$ is selected, or nothing is selected in $\neg A_1 \lor D$ and $\neg A_1$ is $\succ_{lpo}$-maximal with respect to D, and ii) $\sigma = \text{mgu}(A_1 \doteq B_1)$ or

2b. if $n > 1$ and there exists an mgu σ' such that $\sigma' = \text{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$, then $\sigma = \text{mgu}(A_1 \doteq B_1, \ldots, A_m \doteq B_m)$ where $m \leq n$.

3. All premises are variable-disjoint.

Suppose $C_1 = B_1 \lor D_1, \ldots, C_n = B_n \lor D_n$ and $C = \neg A_1 \lor \ldots \lor \neg A_m \lor \ldots \lor \neg A_n \lor D$. Recall that in the **Q-CO**$^{\mathsf{GQ}}$ procedure, one first finds *connected top-variable subclause* C' of C and then disjunctively connected the remainders of side premises, which match literals in C', as a guarded clause. W.l.o.g. suppose $\neg A_1 \vee \neg A_2$ is a connected top-variable subclause of C, and x_1 and x_2 are connected top variables that occurs in $\neg A_1$ and $\neg A_2$, respectively. Suppose x_1 and x_2 pair to s_1 and t_1, respectively, and s_1 and t_1 occur in B_1 and B_2, respectively. Further suppose that s and t, respectively, are compound terms that occur in D_1 and D_2, and u is a compound term in D_1 that is distinct from s. Using the **T-Trans** rule and a fresh predicate symbol P, $(D_1 \vee D_2)\sigma \vee P$ is the obtained guarded clause. Hence our aim is to show all compound terms in $(D_1 \vee D_2)\sigma$ are compatible (as P is a flat literal), that is, s, t and u are compatible.

By **Definition 17** and the fact that x_1 and x_2 are connected top variables, there exists a sequence of top variables $x_1, \ldots x_2$ in C such that each pair of variable in this sequence co-occurs in a literal of C. W.l.o.g. suppose y_1 and y_2 are two variables in this sequence that occur in a literal $\neg A_i$. Hence $\neg A_1 \vee \neg A_2 \vee \neg A_i$ is a connected top-variable subclause of C. By *3.* of **Lemma 4.13**, in B_i, y_1 and y_2 pair either constants or compound terms. By *5.* of **Lemma 4.13**, if either y_1 or y_2 pairs a constant, $(D_1 \vee D_2)\sigma \vee P$ is a flat and ground clause. In this case, the statement immediately holds. Now assume that y_1 and y_2 pair ground compound terms. By the covering property, C_i is a ground compound-term clause, which contradicts the fact that C_i is an aligned guarded clause. Hence, y_1 and y_2 must pair non-ground compound terms. Suppose y_1 and y_2 pairs non-ground compound terms s_2 and t_2, respectively. By **Definition 18**, s_2 and t_2 are compatible. By *2.* of **Lemma 5.2**, $s_2\sigma$ and $t_2\sigma$ are compatible, therefore $y_1\sigma$ and $y_2\sigma$ are compatible. By the fact that y_1 and y_2 are connected to x_1 and x_2, $x_1\sigma$ and $x_2\sigma$ are compatible. By the facts that x_1 pairs to s_1 and x_2 pairs to t_1, $s_1\sigma$ and $t_1\sigma$ are compatible. By the facts that C_1 and C_2 are aligned guarded clauses, s is compatible to s_1 and t is compatible to t_1. Then by *3.* of **Lemma 5.2** and the facts that $s_1\sigma$ and $t_1\sigma$ are compatible, and $s\sigma$ and $t\sigma$ are compatible. By **Definition 18**, s and u are compatible. By *1.* of **Lemma 5.2**, $s\sigma$ and $u\sigma$ are compatible. Hence, $s\sigma$, $u\sigma$ and $t\sigma$ are compatible. Then all compound terms in $(D_1 \vee D_2)\sigma \vee P$ are compatible. □

Now we can give the main result of this section.

Theorem 5.3. *The* ***Q-Ans***$^{\mathsf{GF}}$ *procedure decides satisfiability of the* GQ^- *clausal class.*

Proof. By **Lemma 5.5**, **Theorems 4.5** and **5.2** and the fact that the class of aligned guarded clauses is a strict subset of that of guarded clauses. □

5.3 Back-translating GQ⁻ clausal sets

In this section, we aim to back-translate a GQ⁻ clausal set to a first-order formula. We give our variations of the **Rename**, **Abstract** and **Unsko** rules, based on which we provide the formal procedure that transform a GQ⁻ clausal set into a unique, normal, globally compatible and globally linear clausal set N. In the last step, we back-translate N into a first-order formula.

Since query clauses are free of compound terms, each query clause in GQ⁻ clausal sets N can be straightforwardly unskolemised into a universally quantified first-order formula, without affecting the G⁻ clauses in N. Thus in this section we concentrate on unskolemising clauses, especially compound-term clauses, in the G⁻ clausal class.

Making a GQ⁻ clausal set normal and unique

Normalising GQ⁻ clausal sets

In this section, we give the technique and algorithm that transform GQ⁻ clausal sets to a *normal* and strongly compatible clausal sets.

Recall the definition of *normality* from **Section 3.3**. A GQ⁻ clause is not necessarily normal. For example in the GQ⁻ clause $\neg G(x, a) \lor A(f(x, a), x) \lor B(g(x, a), x)$, constant a occurs in compounds terms $f(x, a)$ and $g(x, a)$.

Constants occurring in compound terms of GQ⁻ clauses are abstracted using

The ConAbs rule

$$\frac{N \cup \{C(a)\}}{N \cup \{C(y) \lor y \not\approx a\}}$$

if the following conditions are satisfied.

1. $C(a)$ is a compound-term GQ⁻ clause.
2. a occurs in compound terms of $C(a)$.
3. y does not occur in $C(y)$.
4. $C(y)$ does not contain a.

Suppose C is a GQ⁻ clause and a is a constant occurring in compound terms of C. By *4.* of the **ConAbs** rule, all occurrences of a are simultaneously abstracted, hence applying the **ConAbs** rule to a GQ⁻ clause produces a *strongly* *compatible* clause. Consider the previous GQ^- clause C as an example, applying the **ConAbs** rule to C derives the strongly compatible clause

$$\neg G(x,y) \vee A(f(x,y),x) \vee B(g(x,y),x) \vee y \not\approx a,$$

rather than $\neg G(x,y) \vee A(f(x,y),x) \vee B(g(x,z),x) \vee y \not\approx a \vee z \not\approx a$.

Algorithm 8 is the procedure that normalises GQ^- clausal sets. In **Algorithm** 8, the AbstractConstant(C) function takes a GQ^- clause C as input, and then is applied to C as follows.

- If the **ConAbs** rule is not applicable to C, then C is returned.
- Otherwise the **ConAbs** rule is recursively applied to C, until no constants occur in compound terms of the **ConAbs** conclusions of C, producing a normal clause C'.

Algorithm 8: Normalising GQ^- clausal sets

Input: A GQ^- clausal set N
Output: A normalised GQ^- clausal set

$N' \leftarrow \emptyset$
foreach *clause* C *in* N **do**
  $C \leftarrow$ AbstractConstant(C)
  $N' \leftarrow N' \cup C$
return N'

Applying the **ConAbs** rule to a GQ^- clause ensures to procedure a *strongly compatible* clause, as the **ConAbs** rule simultaneously pull out all occurrences of a constant.

Lemma 5.6. *Applying* ***Algorithm*** *8 to a* GQ^- *clausal set transforms it to a normal and strongly compatible clausal set* N.

Proof. By the definition of **Algorithm** 8 and **Definition** 18. □

We use the notation GQ^-_n to denote the clausal sets that are obtained by applying **Algorithm** 8 to GQ^- clausal sets. Note that a GQ^-_n clause C may not belong to the GQ^- clausal class (and the GQ clausal class thereof), as C may contain equalities.

Making a GQ^-_n clausal set normal and unique

In this section, we handle duplicate variables that occur in compound terms of GQ^-_n clauses. By handling these duplicate variables, we transform GQ^-_n clausal sets to a normal, *unique*and strongly compatible clausal set.

Recall the definition of *uniqueness* from **Section** 3.3.

Definition 11. *A compound term* $f(t_1, \ldots, t_n)$ *is* unique *if each pair of terms in* $t_1, \ldots, t_n$ *is a pair of distinct variables. A clause* C *is* unique *if every compound term in* C *is unique. A clausal set* N *is* unique *if every compound term in* N *is unique.*

By **Definition** 11, a unique clause C requires that there are no duplicate variables that occur in compound term of C. However, a GQ^-_n clause may not be unique. An example is the GQ^-_n clause

$$C = \neg G(x, x) \vee A(f(x, x), x) \vee A(g(x, x), x).$$

Duplicate variables in compound terms of GQ^-_n clauses are abstracted using

The VarAbs rule

$$\frac{N \cup \{C(f(\ldots, x, \ldots, x, \ldots)\}}{N \cup \{C(f(\ldots, x, \ldots, y, \ldots) \vee y \not\approx x\}}$$

if the following conditions are satisfied.

1. $C(f(\ldots, x, \ldots, x, \ldots))$ is a GQ^-_n clause.
2. y does not occur in $C(f(\ldots, x, \ldots, x, \ldots))$.
3. Let the second x in $f(\ldots, x, \ldots, x, \ldots)$ occur at the position i. Then, in every i position in compound terms of $C(f(\ldots, x, \ldots, x, \ldots))$, x is replaced by y.

Observe that in a GQ^-_n clause C, if a compound term in C contains duplicate variables x, which occurs in positions i and j, then in all compound terms of C, x occurs in positions i and j. By this observation, applying the **VarAbs** rule to a GQ^-_n clause transform it into a unique clause. Consider the previous GQ^-_n clause C as an example. Applying the **VarAbs** rule to C transform it into

$$\neg G(x, y) \vee A(f(x, y), x) \vee A(g(x, y), x) \vee x \not\approx y,$$

which is a unique, normal and strongly compatible clause. Like the **ConAbs** rule, the **VarAbs** rule restricts that one use a common variable to abstract all occurrences of one duplicate variable. For example, applying the **VarAbs** rule to C cannot derive

$$\neg G(x, y) \vee A(f(x, y), x) \vee A(g(x, z), x) \vee x \not\approx y \vee x \not\approx z.$$

Algorithm 9 gives the formal procedure that transforms a GQ^-_n clausal set to a unique, normal and strongly compatible clausal set.

Algorithm 9: Transforming a GQ^-_n clausal set to a unique clausal set

Input: A GQ^-_n clausal set N
Output: A unique clausal set N'

```
N′ ← ∅
foreach clause C in N do
    C ← AbstractVariable(C)
    N′ ← N′ ∪ C
return N′
```

Algorithm 9 aims to remove duplicate variables that occur in the compound terms of GQ^-_n clauses. In **Algorithm** 9, the AbstractVariable(C) function takes a GQ^-_n clause C as input, and then applied to C as follows.

- If the **VarAbs** rule is not applicable to C, then C is returned.
- Otherwise, the **VarAbs** rule is recursively applied to C, until no duplicate variables occur in compound terms of conclusions of C, producing a unique clause C', and then C' is returned.

Lemma 5.7. *Applying* ***Algorithm*** *9 to a* GQ^-_n *clausal set transforms it into a normal, unique and strongly compatible clausal set* N*.*

Proof. By the definition of **Algorithm** 9 and **Lemma** 5.6. □

We use the notation $\mathsf{GQ}^-_\mathsf{nu}$ to denote the normal, unique, locally linear and locally compatible clausal sets that are obtained by applying **Algorithm** 9 to a GQ^-_n clausal set. Moreover, given a GQ^- clausal set N, we use the notation **Q-Abs** to denote the *variable and constant abstraction procedure* of applying the **ConAbs** and the **VarAbs** rules, given as follows.

1. Apply **Algorithm 8** to N, transforming it into a normal, locally linear and locally compatible $\mathsf{GQ}^-_{\mathsf{n}}$ clausal set N_1.
2. Apply **Algorithm 9** to N_1 to transform it into a normal, unique, locally linear and locally compatible $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set N_2.

Preparing a $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set for unskolemisation

In this section, we aim to transform a $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set into a normal, unique, *globally compatible* and *globally linear* clausal set, preparing this clausal set for unskolemisation.

By **Definitions 8**, a globally compatible clausal set N requires that in clauses of N, compound terms that are under the same function symbol share the same argument list. Hence we need to find clauses in N that contain the same function symbols. We use the notions *connected clausal sets* and an *inter-connected clausal set* to formal define this problem.

Definition 19. *Two clauses are* connected clausal sets *if they contain at least one common function symbol, otherwise they are* unconnected. *Two clausal sets are* connected *if they contain at least one common function symbol, otherwise they are* unconnected*.*

A clausal set N is an inter-connected clausal set *if for any pair of clauses C and C' in N, there exists a sequence of clauses $C, \ldots, C'$ in N such that each adjacent pair of clauses in $C, \ldots, C'$ is connected.*

By **Definition 19**, one can partition a $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set N into a clausal set N' containing only flat clauses and inter-connected clausal sets $N_1, \ldots, N_n$ such that each pair of clausal sets in $N_1, \ldots, N_n$ are unconnected. We say $N', N_1, \ldots, N_n$ are *closed clausal sets (with respect to N)*.

The back-translation of a closed $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set consisting of flat clauses, into a first-order formula is easy. Hence we put our focus on unskolemising inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal sets. An inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set has the following nice property:

Lemma 5.8. *Let N be an inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set. Then, all compound terms in N have the same arity.*

Proof. By the fact that $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal sets is strongly compatible (**Lemma 5.7**) and **Definition 19**, compound terms in any pair of connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clauses have the same arity. Then all compound terms in an inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set have the same arity. □

Lemma 5.8 implies that given an inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set N, one can use a sequence of variables to substitute all variable sequences of compound terms in N. Next, we devise the following **VarRe** rule.

In compound terms of inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal sets, the variables arguments are renamed using

The VarRe rule

$$\frac{N \cup \{C(f(x_1, \ldots, x_n))\}}{N \cup \{C(f(y_1, \ldots, y_n)))\}}$$

if the following conditions are satisfied.

1. $N \cup \{C(f(x_1, \ldots, x_n))\}$ is an inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set.
2. All occurrences of $x_1, \ldots, x_n$ in $C(f(x_1, \ldots, x_n))$ are replaced by $y_1, \ldots, y_n$, respectively.
3. $y_1, \ldots, y_n$ do not occur in $N \cup \{C(f(x_1, \ldots, x_n))\}$.

Let N be an inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set. To transform N to a globally compatible clausal set, the **VarRe** rule is applied to all clauses in N. Suppose $f(x_1, \ldots, x_n)$ is a compound term of a clause in N. Using a sequence of fresh variables $y_1, \ldots, y_n$ that does not occur in N, the **VarRe** rule substitutes variables in all clauses of N through $\{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$, so that N can be transformed into a globally compatible clausal set.

We use **ReInt** to denote the procedure of renaming variables of all clauses in inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal sets. By the inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set

$$N = \left\{ \begin{array}{l} \neg G_1(x_1, x_2) \vee A_1(f(x_1, x_2), x_2) \vee x_1 \not\approx a, \\ \neg G_2(x_3, x_4) \vee A_2(f(x_3, x_4), x_3) \vee A_3(g(x_3, x_4), x_3), \\ \neg G_3(x_5, x_6) \vee A_4(g(x_5, x_6), x_5) \vee x_5 \not\approx y \end{array} \right\}$$

(where a is a Skolem constant), we show how the **ReInt** procedure renames variables. The **ReInt** procedure consists of two steps:

1. Find the arity of any compound term in N, which is two. Then introduce a sequence of two fresh variables that do not occur in N. We use y_1, y_2 as

this sequence of fresh variables.

2. For each clause $C(f(x_1, \dots, x_n))$ in N, apply the **VarRe** rule it, substituting $x_1, \dots, x_n$ by $\{x_1 \mapsto y_1, \dots, x_n \mapsto y_n\}$. We obtain the clausal set

$$N' = \left\{ \begin{array}{l} \neg G_1(y_1, y_2) \lor A_1(f(y_1, y_2), y_2) \lor y_1 \not\approx a, \\ \neg G_2(y_1, y_2) \lor A_2(f(y_1, y_2), y_1) \lor A_3(g(y_1, y_2), y_1), \\ \neg G_3(y_1, y_2) \lor A_4(g(y_1, y_2), y_1) \lor y_1 \not\approx y \end{array} \right\}.$$

Then a normal, unique, globally compatible and globally linear clausal set N' is obtained.

Lemma 5.9. *Applying the* ***ReInt*** *procedure to an inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set transforms it into a normal, unique, globally compatible and globally linear clausal set.*

Proof. By **Lemmas** 3.2, 5.7, 5.8 and the definition of the **ReInt** procedure. □

The **ReInt** procedure renames all variables of inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal sets. However to rename the whole of a $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set N, one needs to partition N into closed clausal sets. We use the notation **Q-Rena** to denote the procedure of partitioning $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal sets to closed clausal sets, and then renames all variables of these closed clausal sets. See **Algorithm** 10.

Algorithm 10: Renaming variables of $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal sets

Input: A $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set N
Output: A normal, unique, globally linear and globally compatible clausal set N'

```
N' ← ∅
while there exists compound-term clause in N do
    Find a compound-term clause C in N
    ClosedSet ← FindInt(C, N)
    N ← N \ ClosedSet
    ClosedSet ← Rename(ClosedSet)
    N' ← N' ∪ ClosedSet
N' ← N' ∪ N
return N'
```

Algorithm 10 consists of the following functions:

1. The FindInt(C, N) function takes a $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set N and a compound-term clause C in N as input, outputting the inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set, in which C occurs.

2. The Rename(N) function applies the **ReInt** procedure to rename variables of an inter-connected clausal set N, and outputs a normal, unique, globally compatible and globally linear clausal set.

In **Algorithm** 10, the while-loop in Lines 2–7 iteratively find closed clausal sets in a $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set N, and then removes these closed clausal sets from N. Lines 3–4 first find an arbitrary compound-term clause C in N, and then uses the FindInt(C, N) function to find the C-occurring inter-connected clausal set in N. Then Line 6 rename variables in this inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set. Line 7 uses N' to store the inter-connected clausal sets in N in which variables are renamed. In the last step, Line 8 adds the remaining clauses in N, which are flat clauses, to N'.

Algorithm 11: The FindInt function

Input: A $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set N and a compound-term clause C in N
Output: An inter-connected $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set that C occurs

Function FindInt(C, N)**:**
  FunSym $\leftarrow$ FindFunSymbol(C)
  NewClosedSet $\leftarrow$ FindNewClosedSet(FunSym, N)
  ExistClau $\leftarrow$ NewClosedSet
  NewClau $\leftarrow$ ExistClau $\setminus C$
  while NewClau *is not empty* **do**
    FunSym $\leftarrow$ FindFunSymbol(ExistClau)
    NewClosedSet $\leftarrow$ FindNewClosedSet(FunSym, N)
    NewClaus $\leftarrow$ NewClosedSet $\setminus$ ExistClau
    ExistClau $\leftarrow$ NewClosedSet
  return ExistClau

On the next page **Algorithm** 11 describes the FindInt function, containing the following two functions:

1. The FindFunSymbol(N) function returns all function symbols in the clausal set N.

2. The FindNewClosedSet($\mathtt{F}, N$) takes a set $\mathtt{F}$ of functions symbols and a clausal set N as input, and returns a subset N' of N such that each clause in N' contains at least one function symbol in $\mathtt{F}$.

Lemma 5.10. *Applying the* **Q-Rena** *procedure to a* $\mathsf{GQ}^-_{\mathsf{nu}}$ *clausal set transforms it into a normal, unique, globally compatible and globally linear clausal set.*

Proof. By **Lemma** 5.9 and the definition of **VarRe** procedure. □

We use the notation $\mathsf{GQ}^-_{\mathsf{nucl}}$ to denote the clausal class obtained by applying the **Q-Rena** procedure to the $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal class.

Unskolemising a $\mathsf{GQ}^-_{\mathsf{nucl}}$ clausal set

In this section, we introduce the customised unskolemisation rules for $\mathsf{GQ}^-_{\mathsf{nucl}}$ clausal sets. By these rules, we give the formal procedure that back-translate a $\mathsf{GQ}^-_{\mathsf{nucl}}$ clausal set into a first-order formula.

By **Definition** 19, one can partition a $\mathsf{GQ}^-_{\mathsf{nucl}}$ clausal set into a set of closed $\mathsf{GQ}^-_{\mathsf{nucl}}$ clausal sets, in which are inter-connected $\mathsf{GQ}^-_{\mathsf{nucl}}$ clausal sets and a $\mathsf{GQ}^-_{\mathsf{nucl}}$ clausal set consisting of flat clauses.

If $\mathsf{GQ}^-_{\mathsf{nucl}}$ clausal set N contains only flat clauses, N is unskolemised using

The UnskoOne rule

$$\frac{N \cup \{C_1(x,a), \ldots, C_n(y,b)\}}{N \cup \{\exists z \forall xy (C_1(x,z) \land \ldots \land C_n(y,b))\}}$$

if the following conditions are satisfied.

1. $\{C_1(x,a), \ldots, C_n(y,b)\}$ is a compound-term-free $\mathsf{GQ}^-_{\mathsf{nucl}}$ clausal set,
2. a and b represent Skolem and non-Skolem constants, respectively.
3. z does not occur in $\{C_1(x,a), \ldots, C_n(y,b)\}$.

Consider a compound-term-free $\mathsf{GQ}^-_{\mathsf{nucl}}$ clausal set

$$N = \left\{ \begin{array}{l} \neg G_1(a_1, a_2) \lor \neg A_1(a_1), \\ \neg G_2(y_1, y_2) \lor A_2(a, y_1) \end{array} \right\}$$

where a_1 and a_2 are Skolem constants and a is a non-Skolem constant. The **UnskoOne** rule unskolemises N using the following steps.

1. For each constant a_i in N, introduce an existentially quantified variable and an existential quantification for all occurrences of a_i, obtaining

$$N_1 = \exists z_1 z_2 \left[\begin{array}{l} (\neg G_1(z_1, z_2) \vee \neg A_1(z_1)) \wedge \\ (\neg G_2(y_1, y_2) \vee A_2(a, y_1)) \end{array} \right].$$

2. Add universal quantifications for all free variables in N_1, obtaining a first-order formula

$$\exists z_1 z_2 \forall y_1 y_2 \left[\begin{array}{l} (\neg G_1(z_1, z_2) \vee \neg A_1(z_1)) \wedge \\ (\neg G_2(y_1, y_2) \vee A_2(a, y_1)) \end{array} \right].$$

Next we unskolemise compound-term $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal sets. An inter-connected $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal set is unskolemised using

The UnskoTwo rule

Let N_1 be a $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal set, N be a closed $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal set in N_1, and N' be the set $N_1 \setminus N$. Suppose N is an inter-connected $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal set

$$\left\{ \begin{array}{c} C_1(x_1, \ldots, x_n, f(x_1, \ldots, x_n), z_1, a), \\ \ldots \\ C_n(x_1, \ldots, x_n, g(x_1, \ldots, x_n), z_t, b) \end{array} \right\}$$

where a, b, $x_1, \ldots, x_n$ and $z_1, \ldots, z_t$ represent Skolem constants, non-Skolem constants and variables introduced in the **Q-Rena** and **Q-Abs** procedures, respectively. Let F be the first-order formula

$$\exists y \forall x_1 \ldots x_n \exists y_1 \ldots y_m \forall z_1, \ldots, z_t \left[\begin{array}{c} C_1(x_1, \ldots, x_n, y_1, z_1, y) \wedge \\ \ldots \\ C_n(x_1, \ldots, x_n, y_m, z_t, b) \end{array} \right].$$

Then N is unskolemised to a first-order formula using

$$\frac{N' \cup N}{N' \cup \{F\}}$$

if $y_1, \ldots, y_m$ and y do not occur in $N' \cup N$.

Using the inter-connected $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal set

$$N = \left\{ \begin{array}{l} \neg G_1(y_1, y_2) \vee A_1(f(y_1, y_2), y_2) \vee y_1 \not\approx a, \\ \neg G_2(y_1, y_2) \vee A_2(f(y_1, y_2), y_1) \vee A_3(g(y_1, y_2), y_1), \\ \neg G_3(y_1, y_2) \vee A_4(g(y_1, y_2), y_1) \vee y_1 \not\approx y \end{array} \right\}$$

with a a Skolem constant, the following steps show how the **UnskoTwo** rule unskolemises inter-connected $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal sets.

1. Add existential quantifications and existentially quantified variables to replace Skolem constants in N, obtaining

$$N_1 = \exists z \left[\begin{array}{l} (\neg G_1(y_1, y_2) \vee A_1(f(y_1, y_2), y_2) \vee y_1 \not\approx z) \wedge \\ (\neg G_2(y_1, y_2) \vee A_2(f(y_1, y_2), y_1) \vee A_3(g(y_1, y_2), y_1)) \wedge \\ (\neg G_3(y_1, y_2) \vee A_4(g(y_1, y_2), y_1) \vee y_1 \not\approx y) \end{array} \right].$$

2. Introduce universal quantifications for variables that occur in the compound terms in N_1, obtaining

$$N_2 = \exists z \forall y_1 y_2 \left[\begin{array}{l} (\neg G_1(y_1, y_2) \vee A_1(f(y_1, y_2), y_2) \vee y_1 \not\approx z) \wedge \\ (\neg G_2(y_1, y_2) \vee A_2(f(y_1, y_2), y_1) \vee A_3(g(y_1, y_2), y_1)) \wedge \\ (\neg G_3(y_1, y_2) \vee A_4(g(y_1, y_2), y_1) \vee y_1 \not\approx y) \end{array} \right].$$

3. For each function symbol f_i in N_2, introduce a new existentially quantified variable and a new existential quantification, to replace all occurrences of Skolem compound terms that are under f_i, obtaining

$$N_3 = \exists z \forall y_1 y_2 \exists z_1 z_2 \left[\begin{array}{l} (\neg G_1(y_1, y_2) \vee A_1(z_1, y_2) \vee y_1 \not\approx z) \wedge \\ (\neg G_2(y_1, y_2) \vee A_2(z_1, y_1) \vee A_3(z_2, y_1)) \wedge \\ (\neg G_3(y_1, y_2) \vee A_4(z_2, y_1) \vee y_1 \not\approx y) \end{array} \right].$$

4. Finally, add universal quantifications for free variables in N_3, obtaining

$$F = \exists z \forall y_1 y_2 \exists z_1 z_2 \forall y \left[\begin{array}{l} (\neg G_1(y_1, y_2) \vee A_1(z_1, y_2) \vee y_1 \not\approx a) \wedge \\ (\neg G_2(y_1, y_2) \vee A_2(z_1, y_1) \vee A_3(z_2, y_1)) \wedge \\ (\neg G_3(y_1, y_2) \vee A_4(z_2, x_1) \vee y_1 \not\approx y) \end{array} \right].$$

Lemma 5.11. *Given an inter-connected $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal set N, the **UnskoTwo** rule transforms it into a first-order formula without Skolem symbols.*

Proof. By **Lemma** 5.10 and **Theorem** 3.1. □

We use the notation **Q-Unsko** to denote the procedure of unskolemising a $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal set into a first-order formula. See **Algorithm** 12.

In **Algorithm** 12, the Unsko(N) function takes a closed clausal set N as input. Otherwise the **UnskoTwo** rule is applied to N, outputting a first-order formula. If N is a compound-term-free $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal set, then the **UnskoOne** rule is applied to N, outputting a first-order formula.

Algorithm 12: Unskolemising a $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal set to a formula

Input: A $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ clausal set N
Output: A first-order formula F

```
F ← ∅
foreach closed clausal set N′ in N do
    F₁ ← Unsko(N)
    F ← F₁ ∪ F
return F
```

Lemma 5.12. *Applying the* **Q-Unsko** *procedure to a* $\mathsf{GQ}^{-}_{\mathsf{nucl}}$ *clausal set transforms it to a first-order formula without Skolem symbols, but with equality.*

Proof. By **Lemmas** 5.10–5.11, **Theorem** 3.1 and the definition of the **Q-Unsko** procedure. □

5.4 A decision procedure for rewriting BCQs for GF

By combining all results from the previous sections, we give a decision procedure for saturation-based BCQ rewriting for GF.

We use **Q-Rew**$^{\mathsf{GF}}$ to denote the procedure of rewriting BCQ for GF. See **Algorithm** 13 on the next page.

Algorithm 13 contains the following functions.

- The Q-AnsGF(Σ, q) function takes a set Σ of guarded formulas and a union q of BCQs as input, and then applies the **Q-Ans**$^{\mathsf{GF}}$ procedure to compute the saturation of $\Sigma \cup \{\neg q\}$.

- The Q-Abs(N) function takes a GQ^- clausal set N as input, and applies the **Q-Abs** procedure to N, outputting a unique, normal and strongly compatible clausal set.
- The Q-Rena(N) function takes a $\mathsf{GQ}^-_{\mathsf{nu}}$ clausal set N as input, and then applies the **Q-Rena** procedure to it, returning a normal, unique, globally compatible and globally linear clausal set $\mathsf{GQ}^-_{\mathsf{nucl}}$.
- The Q-Unsko(N) function takes a $\mathsf{GQ}^-_{\mathsf{nucl}}$ clausal set N as input, and then applies the **Q-Unsko** procedure to it, returning a first-order formula without Skolem symbols.

Algorithm 13: The saturation-based BCQ rewriting procedure for GF

Input: A union q of BCQs and a set Σ of guarded formulas
Output: A first-order formula without Skolem symbols

```
N ← Q-AnsGF(Σ, q)
N_1 ← Q-Abs(N)
N_2 ← Q-Rena(N_1)
F ← Q-Unsko(N_2)
return Negated F
```

Lemma 5.13. *The **Q-Rew**$^{\mathsf{GF}}$ procedure preserves logical equivalence.*

Proof. The **Q-Rew**$^{\mathsf{GF}}$ procedure uses variations of the **Rename** rule, the **Abstract** rule and the **Q-Unsko** rule from **Section** 3.3. By **Lemma** 3.3, any variation of these rules are sound and preserve logical equivalence. Hence, the **Q-Rew**$^{\mathsf{GF}}$ procedure preserves logical equivalence. □

Finally, we give a positive answer to **Problem** 5.

Theorem 5.4. *Suppose Σ is a set of formulas in GF, D is a set of ground atoms and q is a union of BCQs. The **Q-Rew**$^{\mathsf{GF}}$ procedure is a decision procedure that negates, and then back-translates the saturated clausal set of $\Sigma \cup \{\neg q\}$ to a (function-free) first-order formula with equality Σ_q such that $\Sigma \cup \mathrm{D} \models q$ if and only if $\mathrm{D} \models \Sigma_q$.*

Proof. By **Theorem** 5.1, the problem of answering BCQs for GF is reduce to that of deciding satisfiability of the GQ^- clausal class. By **Theorem** 5.3 and the fact that the **Q-Ans**$^{\mathsf{GF}}$ procedure is a part of the **Q-Rew**$^{\mathsf{GF}}$ procedure, the

Q-Rew$^{\mathsf{GF}}$ procedure decides satisfiability of the GQ^{-} clausal class. By **Lemmas 5.6, 5.7** and **5.10**, the **Q-Rew$^{\mathsf{GF}}$** procedure ensures to back-translate GQ^{-} clausal sets to a unique, normal, locally linear and locally compatible clausal set N. By **Lemma 5.12**, the **Q-Rew$^{\mathsf{GF}}$** procedure ensures to back-translate N to a first-order formula without Skolem symbols. By **Lemma 5.13**, the **Q-Rew$^{\mathsf{GF}}$** procedure preserves logical equivalence. □

Chapter 6

Querying for LGF and CGF

In this chapter, we investigate the problems of answering and rewriting BCQs for more expressive guarded fragments, namely *the loosely guarded fragment* (LGF) and *the clique guarded fragment* (CGF). We start with investigating the BCQs answering problem for LGF and/or CGF, formally stated as:

Problem 6. *Given a set Σ of formulas in LGF and/or CGF and a union q of BCQs, can a saturation-based procedure decide whether $\Sigma \models q$?*

The next problem is the saturation-based BCQ rewriting problem for LGF and/or CGF, formally stated as:

Problem 7. *Given a set Σ of formulas in LGF and/or CGF, a set D of ground atoms and a union q of BCQs, does there exist a (function-free) first-order formula (with equality) Σ_q that is the negated back-translation of the saturated clausal set of $\Sigma \cup \{\neg q\}$ such that $\Sigma \cup \mathrm{D} \models q$ if and only if $\mathrm{D} \models \Sigma_q$?*

This chapter is organised as follows. We first introduce the clausifications that transform LGF and CGF to the so-called *loosely guarded clauses*. **Section 6.2** gives a new top-variable inference system **T-Inf$^{\mathsf{LGQ}}$**, particularly devised for deciding satisfiability of the *loosely guarded clausal class*, and **Section 6.3** then formally proves the decidability claim. The last section formalises the saturation-based decision procedures for answering and rewriting BCQs for LGF and/or CGF.

6.1 Clausal normal forms of LGF and CGF

In this section, we give our structural transformations that process formulas in LGF and CGF into a proper normal clausal form.

Transforming LGF to the LG clausal class

Recall the definition of LGF from **Section 2.1**.

Definition 2. *The* loosely guarded fragment (LGF) *is a fragment of FOL without function symbols, inductively defined 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 connectives.*
4. *Let F be a loosely guarded formula and $\mathbb{G}$ a conjunction of atoms. Then $\forall \overline{x}(\mathbb{G} \rightarrow F)$ and $\exists \overline{x}(\mathbb{G} \wedge F)$ belong to LGF if*
   - *(a) all free variables of F occur in $\mathbb{G}$, and*
   - *(b) for each variable x in $\overline{x}$ and each variable y occurring in $\mathbb{G}$ that is distinct from x, x and y co-occur in an atom of $\mathbb{G}$.*

The **Trans**$^{\mathsf{GF}}$ process (from **Section 4.1**) was originally devised for transforming guarded formulas, however this process also is sufficient to transform a loosely guarded formula to a set of *loosely guarded clauses*. We use the loosely guarded formula

$$F = \exists x_2(A_1(x_1, x_2) \wedge B(x_2) \wedge \forall x_3((A_1(x_1, x_3) \wedge A_1(x_3, x_2)) \rightarrow \exists x_4 A_2(x_4, x_2)))$$

as a sample to show how the **Trans**$^{\mathsf{GF}}$ process is applied, given as follows.

1. Add existential quantifiers to all free variables of F, and by the **NNF** rules, transforming F to negation normal form, obtaining

$$F_1 = \left[\begin{array}{ll} \exists x_1 x_2(& A_1(x_1, x_2) \wedge B(x_2) \wedge \forall x_3(\\ & \neg A_1(x_1, x_3) \vee \neg A_1(x_3, x_2) \vee \exists x_4 A_2(x_4, x_2)) \quad) \end{array} \right].$$

2. By introducing predicate symbols P_i (and respective literals $P_i(\cdots)$), applying the **Trans** rules for each universally quantified subformula of F_1.

Then we obtain

$$F_2 = \begin{bmatrix} \exists x_1 x_2 (A_1(x_1,x_2) \wedge B(x_2) \wedge P(x_1,x_2)) \wedge \\ \forall x_1 x_2 x_3 (\neg P(x_1,x_2) \vee \neg A_1(x_1,x_3) \vee \neg A_1(x_3,x_2) \vee \exists x_4 A_2(x_4,x_2))) \end{bmatrix}.$$

We say that

- $\exists x_1 x_2 (A_1(x_1,x_2) \wedge B(x_2) \wedge P(x_1,x_2))$ is the *replacing formula of* F_1, and
- $\forall x_1 x_2 x_3 (\neg P(x_1,x_2) \vee \neg A_1(x_1,x_3) \vee \neg A_1(x_3,x_2) \vee \exists x_4 A_2(x_4,x_2)))$ is the *definition formula of* P.

3. Transform each immediate subformula of F_2 to prenex normal form, and then applying the **Skolem** rule to the resulting formula. By introducing Skolem constants a, b and a Skolem function $f(x_1, x_2, x_3)$, we obtain

$$F_3 = \begin{bmatrix} A_1(a,b) \wedge B(b) \wedge P_1(a,b) \wedge \\ \neg P(x_1,x_2) \vee \neg A_1(x_1,x_3) \vee \neg A_1(x_3,x_2) \vee A_2(f(x_1,x_2,x_3),x_2)) \end{bmatrix}.$$

4. Drop universal quantifiers, and then transform F_3 to conjunctive normal form, obtaining a set of *loosely guarded clauses*

$$\left\{ \begin{array}{l} A_1(a,b),\ B(b),\ P_1(a,b), \\ \neg P(x_1,x_2) \vee \neg A_1(x_1,x_3) \vee \neg A_1(x_3,x_2) \vee A_2(f(x_1,x_2,x_3),x_2)) \end{array} \right\}$$

The formal definition of the loosely guarded clauses is given as follows.

Definition 20. *A* loosely guarded clause (LG clause) C *is a simple and covering clause, satisfying the following conditions:*

1. *C is either ground, or*
2. *C contains a negative flat subclause $\neg G_1 \vee \ldots \vee \neg G_n$ such that each variable pair in C co-occurs in a literal of $\neg G_1 \vee \ldots \vee \neg G_n$.*

In 2. of **Definition 20**, the negative flat subclause $\neg G_1 \vee \ldots \vee \neg G_n$ is called the *loose guard* of the LG clause C. The class of LG clauses strictly extends that of guarded clauses, since given an LG clause C, one can restrict the loose guard in C to be single literal to obtain a guarded clause, but not vice-versa.

Consider the clauses

$$C_1 = \neg A_1(x,y) \vee \neg A_2(y,z) \vee \neg A_3(z,x),$$
$$C_2 = \neg B_1(x,y,a) \vee \neg B_2(y,z,b) \vee \neg B_3(z,x,w).$$

The clause C_1 is an LG clause (and a query clause), but C_2 is not, as w and y do not co-occur in any negative flat literal, which does not satisfy *2.* of **Definition 20**.

Lemma 6.1. *Applying the* **Trans$^{\mathsf{GF}}$** *process to a loosely guarded formula transforms it into a set of* LG *clauses.*

Proof. We prove that the **Trans$^{\mathsf{GF}}$**process transforms a loosely guarded formula to a set of LG clauses. Suppose F is a loosely guarded formula. In the **Trans$^{\mathsf{GF}}$** process, 1.–2. use new predicate symbols (and literals) to rename universally quantified formulas in F. W.l.o.g., suppose P is the newly introduced predicate symbol, F_1 is the definition formula of P, and F' is the replacing formula of F. Now we show that 3.–4. transform F_1 and F' into LG clauses. F' is an existentially quantified sentence, hence, skolemising F' transforms it into (a set of) flat ground clauses (if conjunctions occur in F'), which are LG clauses. F_1 can be represented as

$$\forall \overline{x}(P(\overline{x}) \rightarrow \forall \overline{y}(\mathbb{G}(\overline{x}, \overline{y}) \rightarrow \phi(\overline{y})))$$

where i) $\mathbb{G}(\overline{x}, \overline{y})$ is in the form of $G_1 \wedge \ldots \wedge G_n$ where each variable in $\overline{y}$ and each variable in $\overline{x} \cup \overline{y}$ co-occur in an atom of $G_1 \wedge \ldots \wedge G_n$ (by *4b.* of **Definition 2**), ii) $\phi(\overline{y})$ consists of atoms and existentially quantified formulas that are connected by Boolean connectives. By 4. of the **Trans$^{\mathsf{GF}}$** process, F_1 is simplified as

$$F_1' = \forall \overline{xy}(\neg P(\overline{x}) \vee \neg G_1(\ldots) \vee \ldots \vee \neg G_n(\ldots) \vee \phi(\overline{y})).$$

Suppose C is the clause obtained from F_1'. 1) $\neg P \vee \neg G_1 \vee \ldots \vee \neg G_n$ is a loose guard of C, since each pair of distinct variables in $\overline{x} \cup \overline{y}$ co-occurs in a literal of $\neg P \vee \neg G_1 \vee \ldots \vee \neg G_n$. 2) for any existential quantified variable z in $\phi(\overline{y})$, z is Skolemised into a compound term that contains $\overline{xy}$. This ensures that any compound term in C shares the same variable set as C. 3) Since F_1' is contains no function symbol, C contains non-nested compound terms. By 1)–3), C is simple, covering and contains a loose guard. Hence, it is an LG clause. □

Recall the fact that **Trans**$^{\mathsf{GF}}$ process transforms a union of BCQs to a set of *query clauses* (from **Section 4.1**). We use the notation LGQ to denote the class of LG clauses and query clauses.

Theorem 6.1. *The **Trans**$^{\mathsf{GF}}$ process reduces the problem of BCQ answering for LGF to that of deciding satisfiability of the LGQ clausal class.*

Proof. By **Theorem 4.1** and **Lemma 6.1**. □

Transforming CGF to the LG clausal class

In this section, we give a customised novel clausification process that transforms clique guarded formulas to an LG clausal set.

Recall the definition of CGF from **Section 2.1**.

Definition 3. *The* clique guarded fragment (CGF) *is a fragment of FOL without function symbols, inductively defined as follows:*

1. *$\top$ and $\bot$ belong to CGF.*
2. *If A is an atom, then A belongs to CGF.*
3. *CGF is closed under Boolean connectives.*
4. *Let F be a clique guarded formula and $\mathbb{G}(\overline{x}, \overline{y})$ a conjunction of atoms. Then $\forall \overline{z}(\exists \overline{x} \mathbb{G}(\overline{x}, \overline{y}) \rightarrow F)$ and $\exists \overline{z}(\exists \overline{x} \mathbb{G}(\overline{x}, \overline{y}) \wedge F)$ belong to CGF, if*
   (a) *all free variables of F occur in $\overline{y}$, and*
   (b) *each variable in $\overline{x}$ occurs in only one atom of $\mathbb{G}(\overline{x}, \overline{y})$, and*
   (c) *for each variable z in $\overline{z}$ and each variable y occurring in $\mathbb{G}(\overline{x}, \overline{y})$ that is distinct from z, z and y co-occur in an atom of $\exists \overline{x} \mathbb{G}(\overline{x}, \overline{y})$.*

Unlike the notion of the loose guard $\mathbb{G}$ in *4.* of **Definition 2**, the clique guard $\exists \overline{x} \mathbb{G}(\overline{x}, \overline{y})$ contains existential quantifications with existentially quantified variables $\overline{x}$. Because of the occurrence of these arbitrary existential quantifications, the clausal normal form of clique guarded formulas cannot be easily defined. For example, consider the clique guard formula

$$F' = \forall x_1 x_2 x_3 (\exists x_4 x_5 (A_1(x_1, x_3, x_4) \wedge A_2(x_2, x_3, x_5) \wedge A_3(x_1, x_2)) \rightarrow \exists x_6 B(x_1, x_6)).$$

Using a Skolem function symbol f, clausifying F' (for example, by the **Trans**$^{\mathsf{GF}}$ process) transforms it into

$$C' = \neg A_1(x_1, x_3, x_4) \vee \neg A_2(x_2, x_3, x_5) \vee \neg A_3(x_1, x_2) \vee B(x_1, f(x_1, x_2, x_3, x_4, x_5)),$$

which is neither an LG clause nor a guarded clause. Observe that although C' has the covering property, variables in it do not have the variable co-occurrence property. Nonetheless in C', x_1, x_2 and x_3 have this variable co-occurrence property. Based on this observation, we realise that one can use the **Miniscoping** and the **Trans** rules to handle the existential quantifications in F'. For example, by the **Miniscoping** rules, F' is transformed into

$$F'_1 = \forall x_1 x_2 x_3 (\exists x_4 A_1(x_1, x_3, x_4) \wedge \exists x_5 A_2(x_2, x_3, x_5) \wedge A_3(x_1, x_2) \rightarrow \exists x_6 B(x_1, x_6)),$$

and then by the **Trans** rules, one can abstract existential quantified formulas in the clique guard of F'_1. Using new predicate symbols P'_1 and P'_2 for $\exists x_4 A_1(x_1, x_3, x_4)$ and $\exists x_5 A_2(x_2, x_3, x_5)$, respectively, F'_1 is transformed into

$$F'_2 = \begin{bmatrix} \forall x_1 x_2 x_3 (P'_1(x_1, x_3) \wedge P'_2(x_2, x_3) \wedge A_3(x_1, x_2) \rightarrow \exists x_6 B(x_1, x_6)) & \wedge \\ \forall x_1 x_3 (\exists x_4 A_1(x_1, x_3, x_4) \rightarrow P'_1(x_1, x_3)) & \wedge \\ \forall x_2 x_3 (\exists x_5 A_2(x_2, x_3, x_5) \rightarrow P'_2(x_2, x_3)) & \end{bmatrix}.$$

Finally by the **Skolem** rules (using a new Skolem function symbol g) and then the **CNF** rules, F'_2 is transformed into a set of *loosely guarded clauses*:

$$\begin{aligned} &\neg P'_1(x_1, x_3) \vee \neg P'_2(x_2, x_3) \vee \neg A_3(x_1, x_2) \vee B(x_1, g(x_1, x_2, x_3)), \\ &\neg A_1(x_1, x_3, x_4) \vee \neg P'_1(x_1, x_3), \\ &\neg A_2(x_2, x_3, x_5) \vee \neg P'_2(x_2, x_3). \end{aligned}$$

We use the notation **Trans**$^{\textsf{CGF}}$ to denote the structural transformation for clique guarded formulas and a union of BCQs. Like the **Trans**$^{\textsf{GF}}$ process, the **Trans**$^{\textsf{CGF}}$ process first negate a union of BCQs to obtains a set of *query clauses*. The next step of the **Trans**$^{\textsf{CGF}}$ process is computing clausal normal forms of clique guarded formulas. We use the clique guarded formula

$$F = \begin{bmatrix} \forall x_1 x_2 (& G(x_1, x_2) \rightarrow \forall x_3 (& \\ & \exists x_4 x_5 (A(x_1, x_3, x_4) \wedge B(x_2, x_3, x_5)) \rightarrow & \\ & \exists x_6 D(x_1, x_6)) &) \end{bmatrix}$$

to elucidate the application of the **Trans**$^{\textsf{CGF}}$ process to clique guarded formulas, given as follows.

1. Add existential quantification for free variables in F, and then apply the **Miniscoping** rules to clique guards in F, obtaining

$$F_1 = \left[\begin{array}{lll} \forall x_1 x_2(& G(x_1, x_2) \to \forall x_3(& \\ & \exists x_4 A(x_1, x_3, x_4) \land \exists x_5 B(x_2, x_3, x_5) \to & \\ & \exists x_6 D(x_1, x_6)) &) \end{array} \right].$$

2. By the **NNF** rules, transform F_1 to negation normal, obtaining

$$F_2 = \left[\begin{array}{lll} \forall x_1 x_2(& \neg G(x_1, x_2) \lor \forall x_3(& \\ & \forall x_4 \neg A(x_1, x_3, x_4) \lor \forall x_5 \neg B(x_2, x_3, x_5) \lor & \\ & \exists x_6 D(x_1, x_6)) &) \end{array} \right].$$

3. Then the **Trans** rules are used as follows. i) For each universally quantified atomic formula that occurs in the clique guard of F_2', we introduce a fresh predicate symbol P_i (and respective literals $\neg P_i(\cdots)$), and ii) for the rest of universally quantified formulas of F_2', we introduce fresh predicate symbols P_j (and respective literals $P_j(\cdots)$). Then from F_2, we obtain F_3, representing as

$$\left[\begin{array}{l} p \\ (\neg p \lor \forall x_1 x_2(\neg G(x_1, x_2) \lor P_1(x_1, x_2))) \land \\ \forall x_1 x_3(P_2(x_1, x_3) \lor \forall x_4 \neg A(x_1, x_3, x_4)) \land \\ \forall x_2 x_3(P_3(x_2, x_3) \lor \forall x_5 \neg B(x_2, x_3, x_5)) \land \\ \forall x_1 x_2(\neg P_1(x_1, x_2) \lor \forall x_3(\neg P_2(x_1, x_3) \lor \neg P_3(x_2, x_3) \lor \exists x_6 D(x_1, x_6))) \end{array} \right].$$

   In F_3, we say that

$$\neg p \lor \forall x_1 x_2(\neg G(x_1, x_2) \lor P_1(x_1, x_2)),$$
$$\forall x_1 x_2(\neg P_1(x_1, x_2) \lor \forall x_3(\neg P_2(x_1, x_3) \lor \neg P_3(x_2, x_3) \lor \exists x_6 D(x_1, x_6))),$$
$$\forall x_1 x_3(P_2(x_1, x_3) \lor \forall x_4 \neg A(x_1, x_3, x_4)),$$
$$\forall x_2 x_3(P_3(x_2, x_3) \lor \forall x_5 \neg B(x_2, x_3, x_5)),$$

   are the *definition formulas of* p, P_1, P_2 and P_3, respectively, and p is the *replacing formula of* F_2.
4. Transform each immediate subformula of F_3 (connecting by conjunctions) to prenex normal form, and then apply **Skolem** rule. Using a Skolem

function symbol f, F_3 is transformed into F_4, presenting as

$$\begin{bmatrix} p \\ (\neg p \vee \forall x_1 x_2(\neg G(x_1,x_2) \vee P_1(x_1,x_2)))\wedge \\ \forall x_1 x_3 x_4(P_2(x_1,x_3) \vee \neg A(x_1,x_3,x_4))\wedge \\ \forall x_2 x_3 x_5(P_3(x_2,x_3) \vee \neg B(x_2,x_3,x_5))\wedge \\ \forall x_1 x_2 x_3(\neg P_1(x_1,x_2) \vee \neg P_2(x_1,x_3) \vee \neg P_3(x_2,x_3) \vee D(x_1,f(x_1,x_2,x_3))) \end{bmatrix}.$$

5. Drop universal quantifiers of F_4, and then by the **CNF** rules, F_4 is transformed to a set of *LG clauses*

$$\left\{ \begin{array}{ll} p, & \neg p \vee \neg G(x_1,x_2) \vee P_1(x_1,x_2), \\ P_2(x_1,x_3) \vee \neg A(x_1,x_3,x_4), & P_3(x_2,x_3) \vee \neg B(x_2,x_3,x_5), \\ \multicolumn{2}{l}{\neg P_1(x_1,x_2) \vee \neg P_2(x_1,x_3) \vee \neg P_3(x_2,x_3) \vee D(x_1,f(x_1,x_2,x_3))} \end{array} \right\}.$$

Lemma 6.2. *The* ***Trans***$^{\mathsf{CGF}}$ *process reduces the problem of deciding satisfiability of clique guarded formulas to that of deciding satisfiability of the LG clausal class.*

Proof. We show that the **Trans**$^{\mathsf{CGF}}$ process transforms a clique guarded formula to a set of LG clauses. Suppose F is a clique guarded formula. In *1.–3.* of the **Trans**$^{\mathsf{CGF}}$ process, universal quantified subformula in F are abstracted. W.l.o.g. we use a new predicate symbol P_1 (and respective literal $\neg P_1(\cdots)$) to abstract universally quantified formulas that occur in the clique guard of F, and we use P_2 (and respective literal $P_2(\cdots)$) to abstract the rest of universally quantified formulas in F. Suppose F_1 and F_2 are the definition formulas of P_1 and P_2, respectively, and F' is the replacing formula of F. Now we show that by 4.–5. of the **Trans**$^{\mathsf{CGF}}$ process, F_1, F_2 and F' are transformed to a set of LG clauses. Suppose $\forall \overline{y} \neg L(\overline{x}, \overline{y})$ is an atomic formula in a clique guard. Then F_1 can be represented in the form of $\forall \overline{x}(P_1(\overline{x}) \vee \forall \overline{y} \neg L(\overline{x}, \overline{y}))$. 4.–5. of the **Trans**$^{\mathsf{CGF}}$ process transforms F_1 to $P_1(\overline{x}) \vee \neg L(\overline{x}, \overline{y})$, which immediately is an LG clause. F_2 can be presented as

$$\forall \overline{x}\overline{y}(\neg P_2(\overline{x}) \vee \neg G_1(\ldots) \vee \ldots \vee \neg G_n(\ldots) \vee \phi(\overline{y})),$$

where i) $\phi(\overline{y})$ is a formula of atoms and existentially quantified formulas that are connected by Boolean connectives, and ii) each pair of distinct variables in $\overline{x} \cup \overline{y}$ co-occurs in a literal of $\neg P_2(\overline{x}) \vee \neg G_1(\ldots) \vee \ldots \vee \neg G_n(\ldots)$. Note that $\phi(\overline{y})$ contains no universal quantifications. By **Lemma 6.1**, 4.–5. transform

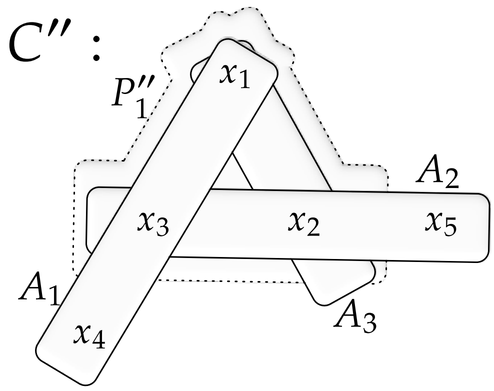

Figure 6.1: The hypergraphs associated with C''

F_2 to a set of LG clauses. Because F' is an existentially quantified sentence without function symbols, skolemising F' transform it into (a set of) flat ground clauses (if conjunctions occur in F'), which are LG clauses. □

To handle existential quantifications in the clique guards, one can also use the **Sep** rule. Recall the clique guarded formula

$$F' = \forall x_1x_2x_3(\exists x_4x_5(A_1(x_1,x_3,x_4) \land A_2(x_2,x_3,x_5) \land A_3(x_1,x_2)) \longrightarrow \exists x_6B(x_1,x_6))$$

from the previous example. Using a new predicate symbol P_1'' and the **Trans** rules, we abstract the clique guard in F', transforming F' into $F_1'' \land F_2''$ where

$$F_1'' = \forall x_1x_2x_3(P_1''(x_1,x_2,x_3) \longrightarrow \exists x_6B(x_1,x_6)) \text{ and}$$
$$F_2'' = \forall x_1x_2x_3(\exists x_4x_5(A_1(x_1,x_3,x_4) \land A_2(x_2,x_3,x_5) \land A_3(x_1,x_2)) \longrightarrow P_1''(x_1,x_2,x_3)).$$

Using the **Skolem** rule and a Skolem symbol g, the subformula F_1'' is transformed into a *(loosely) guarded clause* $\neg P_1''(x_1,x_2,x_3) \lor B(x_1,g(x_1,x_2,x_3))$. Since in F_2'', the clique guard

$$\exists x_4x_5(A_1(x_1,x_3,x_4) \land A_2(x_2,x_3,x_5) \land A_3(x_1,x_2))$$

occurs negatively, F_2'' is transformed into

$$C'' = \neg A_1(x_1,x_3,x_4) \lor \neg A_2(x_2,x_3,x_5) \lor \neg A_3(x_1,x_2) \lor P_1''(x_1,x_2,x_3).$$

By presenting C'' in its associated hypergraph (see Figure 6.1), we realise that one can use the **Sep** rule to 'cut off branches' of C'', transforming C'' to LG clauses. By introducing fresh predicate symbols P_2'' and P_3'', applying the **Sep** rule to C'' separates it into *LG clauses*

$$\neg P_2''(x_1, x_3) \vee \neg P_3''(x_2, x_3) \vee \neg A_3(x_1, x_2) \vee P_1''(x_1, x_2, x_3),$$
$$\neg A_1(x_1, x_3, x_4) \vee P_2''(x_1, x_3), \neg A_2(x_2, x_3, x_5) \vee P_3''(x_2, x_3).$$

To sum up, by the previous process the clique guarded formula F' is transformed into a set of LG clauses

$$\neg P_1''(x_1, x_2, x_3) \vee B(x_1, g(x_1, x_2, x_3)),$$
$$\neg P_2''(x_1, x_3) \vee \neg P_3''(x_2, x_3) \vee \neg A_3(x_1, x_2) \vee P_1''(x_1, x_2, x_3),$$
$$\neg A_1(x_1, x_3, x_4) \vee P_2''(x_1, x_3), \ \neg A_2(x_2, x_3, x_5) \vee P_3''(x_2, x_3).$$

This sample process shows that in the **Trans**$^{\mathsf{CGF}}$ process, one can use i) the applications of the **Trans** and the **Sep** rules, instead of ii) the applications of the **Miniscoping** and the **Trans** rules, to handle clique guards. However ii) needs fewer new symbols and produces fewer clauses. For example, given F' from the previous process, i) transforms it into four clauses with three new predicate symbols, whereas ii) requires two new predicate symbols and produces three clauses. The reason for this fact is that i) needs an additional predicate symbol for the whole of clique guard while ii) does not. Hence we use the current form of the **Trans**$^{\mathsf{CGF}}$ process, which is also more intuitive.

Now we give the result of structural transformation for CGF and BCQs.

Theorem 6.2. *The **Trans**$^{\mathsf{CGF}}$ process reduces the problem of BCQ answering for CGF to that of deciding satisfiability of the LGQ clausal class.*

Proof. By **Theorem** 4.1 and **Lemma** 6.2. □

6.2 The top-variable refinement for the LGQ clausal class

In this section, we give the top-variable inference system **T-Inf**$^{\mathsf{LGQ}}$, which is an extension of the **T-Inf**$^{\mathsf{GQ}}$ system from **Section** 4.3. The **T-Inf**$^{\mathsf{LGQ}}$ system is specially devised for deciding satisfiability of the LGQ clausal class.

The **T-Inf**$^{\mathsf{LGQ}}$ system consists of the same rules as the **T-Inf**$^{\mathsf{GQ}}$ system, but with a new resolution refinement **T-Ref**$^{\mathsf{LGQ}}$. The **T-Ref**$^{\mathsf{LGQ}}$ refinement consists of the same functions and the same resolution refinement as the ones in the **T-Ref**$^{\mathsf{GQ}}$ refinement from **Section** 4.3, however this **T-Ref**$^{\mathsf{LGQ}}$ considers clauses that are loosely guarded, but not guarded. Recall that in the **T-Ref**$^{\mathsf{GQ}}$ refinement, we can use any admissible orderings with a precedence in which function symbols are larger than constant, which are larger than predicate symbols. Here a lexicographic path ordering $\succ_{lpo}$ is used as an example. The application of the **T-Ref**$^{\mathsf{LGQ}}$ refinement to LGQ clauses is given in **Algorithm** 14.

Algorithm 14: Determining the (**P-Res**) eligible literals for LGQ clauses

Input: An LGQ clausal set N and a clause C in N
Output: The eligible literals or the **P-Res** eligible literals (with respect to a **Res** inference) in C

1 **if** *C is a ground clause* **then**
2   **return** Max(C)
3 **else if** *C has negatively occurring compound-term literals* **then**
4   **return** SelectNC(C)
5 **else if** *C has positively occurring compound-term literals* **then**
6   **return** Max(C)
7 **else** **return** PResT(N, C)

Recall that **Algorithm** 1 is the procedure that determines the (**P-Res**) eligible literals for GQ clauses. In 1–6 of **Algorithm** 14, we use the same strategy as the one for determining the (**P-Res**) eligible literals for the GQ clauses (given in 1–6 of **Algorithm** 1), that is i) for ground clauses C, the $\succ_{lpo}$-maximal literal with respect to C is eligible, and ii) for compound-term clauses, one of its compound-term literal is eligible. Unlike **Algorithm** 1, in Line 7 **Algorithm** 14 uses the PResT function to determine the **P-Res** eligible literals for *flat guarded clauses* (and flat LG clauses). This means one needs to perform a top-variable resolution inference step on a flat guarded clauses (as the main premise) and LGQ clauses (as the side premises), which causes the overhead of computing top-variable literals of flat guarded clauses C (with respect to a top-variable inference step). By the SelectG function in **Algorithm** 1, one can select the guard in C (that contains all variables of C) instead, so that only the binary form of the **P-Res** rule is needed for C. We keep the current form of **Algorithm** 14 for its compactness.

By the covering property of LGQ clauses, an *a priori checking*, as well as an *a posteriori checking*, can be used in applying the **T-Inf**$^{\mathsf{LGQ}}$ system to LGQ clauses. This is formally stated as:

Lemma 6.3. *Under the restrictions of the* ***T-Ref***$^{\mathsf{LGQ}}$ *refinement, if an eligible literal L is (strictly) $\succeq_{lpo}$-maximal in an* LGQ *clause C, then $L\sigma$ is (strictly) $\succeq_{lpo}$-maximal in $C\sigma$, for any substitution σ.*

Proof. By **Lemma** 4.6 and the fact that LGQ clauses are covering clauses. □

The main result of this section is given as follows.

Theorem 6.3. *The* ***T-Inf***$^{\mathsf{LGQ}}$ *system is sound and refutational complete for general first-order clausal logic.*

Proof. By **Theorem** 4.3 and the facts that the **T-Inf**$^{\mathsf{LGQ}}$ system consists of the same rules as the **T-Inf**$^{\mathsf{GQ}}$ system, and these rules are refined by admissible orderings with selection functions and a particular form of the **P-Res** rule. □

6.3 Deciding the LGQ clausal class

In this section, we first formally prove that the **T-Inf**$^{\mathsf{LGQ}}$ system decides satisfiability of the LG clausal class, and then investigate inference steps of query clauses and LG clauses.

Deciding satisfiability of the LG clausal class

In this section, we show that the **T-Inf**$^{\mathsf{LGQ}}$ system decides satisfiability of the LG clausal class. By the facts that the LG clausal class extends the guarded clausal class by replacing guards by loose guards and the **T-Inf**$^{\mathsf{LGQ}}$ system extends the **T-Inf**$^{\mathsf{GQ}}$ system correspondingly, the result of this section heavily rely on the lemmas established in **Section** 6.3. In particular we focus on the cases when loose guards, rather than guards, are the (**P-Res**) eligible literals in applications of the rules from the **T-Inf**$^{\mathsf{LGQ}}$ system.

First we show that the **T-Ref**$^{\mathsf{LGQ}}$ refinement ensures that in an LG clause C, the eligible literals or the **P-Res** eligible literals (with respect to a **Res** inference) contain the same variable set as C. This is formally stated as:

Lemma 6.4. *Under the restrictions of the* ***T-Ref***$^{\mathsf{LGQ}}$ *refinement, the eligible literals or the* ***P-Res*** *eligible literals (with respect to a* ***Res*** *inference) in an* LG *clause* C *share the same variable set as* C*.*

Proof. By **Algorithm 14**, we distinguish three cases of C:

Lines 1–2: When C is ground the statement trivially holds.

Lines 3–6: Suppose C is a compound-term LG clause and L is the eligible literal in C. By **Lemma 4.5** (if L is positive) and the definition of the SelectNC function (if L is negative), L is a compound-term literal. By the covering property, $\text{var}(L) = \text{var}(C)$.

Lines 7: Suppose C is a flat LG clause and $\mathbb{L}$ are the **P-Res** eligible literals (top-variable literals) in C. Assume x is a top variable in C. By 2. of **Definition 20** and the definition of top-variable literals, x co-occurs with all other variables of C in $\mathbb{L}$, therefore $\text{var}(\mathbb{L}) = \text{var}(C)$. □

The **T-Ref**$^{\mathsf{LGQ}}$ refinement ensures that in an LG clause, the deepest literal in it is eligible. Specifically Lines 3–6 of **Algorithm 14** ensure that in a non-ground compound-term LG clause, at least one of its compound-term literals is eligible.

Next, we give the pairing properties in the applications of the top-variable resolution rule to a flat clause and LG clauses.

Lemma 6.5. *In an application of the* ***P-Res*** *rule, endowed with the* ***T-Ref***$^{\mathsf{LGQ}}$ *refinement, to a flat clause as a main premise and* LG *clauses as side premises, the following conditions hold.*

1. *In the main premise, top variables pair constants or compound terms, and non-top variables pair constants or variables.*
2. *In the eligible literals of side premises, compound terms pair top variables, and variables or constants pair non-top variables.*
3. *In the main premise, top variables x are unified with either constants or the compound term pairing x (modulo variables substituted with either variables or constants), and non-top variables are unified with either constants or variables.*
4. *In the side premises, variables are unified with either constants or variables.*

5. *Let a top variable x pair a constant. Then in the main premise, all negative literals are the top-variable literals and all variables are unified with constants.*

Proof. By **Lemma** 4.13. □

Lemma 6.6. *In an application of the **P-Res** rule, endowed with the **T-Ref**$^{\mathsf{LGQ}}$ refinement, to a flat clause as the main premise and LG clauses as the side premises, the **P-Res** resolvent is no deeper than its premises.*

Proof. By *3.–4.* in **Lemma** 6.5. □

Now we investigate the applications of the **Fact** and **P-Res** rules to LG clauses, starting with the application of the **Fact** rule.

Lemma 6.7. *In the application of the **Fact** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}}$ refinement) to LG clauses, the factor of an LG clause is an LG clause.*

Proof. By adapting 'guards' to 'loose guards' in the proof of **Lemma** 4.15. □

We then discuss the resolvents of applying the top-variable resolution rule to the LG clauses.

Lemma 6.8. *In the application of the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}}$ refinement) to LG clauses, the resolvents of LG clauses are LG clauses.*

Proof. By **Algorithm** 14, we consider the case when the top-variable technique is used in the **P-Res** rule. For the rest of cases of performing inference steps on LG clauses, their results can be obtained by adapting 'guards' to 'loose guards' in the proof of **Lemma** 4.16.

Assume the side premises are LG clauses $C_1 = B_1 \vee D_1, \ldots, C_n = B_n \vee D_n$, the main premise is an LG clause $C = \neg A_1 \vee \ldots \vee \neg A_m \vee \ldots \vee \neg A_n \vee D$ and the resolvent is $C' = (D_1 \vee \ldots \vee D_m \vee \neg A_{m+1} \vee \ldots \vee \neg A_n \vee D)\sigma$ with σ an mgu such that $\sigma = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_m \doteq B_m)$. By Line 7 in **Algorithm** 14, C is a non-ground flat LG clause and $\neg A_1 \vee \ldots \vee \neg A_m$ is a top-variable subclause of C. By *3.–4.* in **Lemma** 6.5, the mgu σ substitutes variables in $D_1, \ldots, D_m$ and $\neg A_{m+1} \vee \ldots \vee \neg A_n \vee D$ with either variables or constants, therefore C' is simple. Next we prove that C' is covering and contains a loose guard. Suppose $x_1, \ldots, x_{m'}$ are the set of top variables in C. By *3.* in **Lemma** 6.5, any variable x_i in $x_1, \ldots, x_{m'}$ is substituted by either a compound term or a constant that x_i pairs. First suppose x_i pairs a constant. By **Lemma** 6.4 and *5.* of **Lemma** 6.5,

C' is a flat and ground clause, therefore C' is an LG clause. Next, suppose x_i pairs a compound term. W.l.o.g. further suppose x_i and x_j co-occur in a literal $\neg A_t$ of $\neg A_1 \vee \ldots \vee \neg A_m$. By the facts that x_i and x_j are top variables and x_i pairs a compound term, x_j pairs a compound term as well. Then $x_i\sigma$ and $x_j\sigma$ are compound terms. Suppose $C_t = B_t \vee D_i$ is the side premise that A_t pairs B_t. By the covering property, $\text{var}(x_i\sigma) = \text{var}(x_j\sigma) = \text{var}(A_t\sigma) = \text{var}(B_t\sigma)$. By *2.* of **Definition 20** (the variable co-occurrence property), $\text{var}(x_1\sigma) = \ldots = \text{var}(x_{m'}\sigma) = \text{var}((\neg A_1 \vee \ldots \vee \neg A_m)\sigma) = \text{var}(B_t\sigma)$. By *2.* of **Definition 20** and **Lemma 6.4**, $\text{var}(C) = \text{var}(\neg A_1 \vee \ldots \vee \neg A_m)$. Thus $\text{var}(C\sigma) = \text{var}(x_1\sigma) = \ldots = \text{var}(x_{m'}\sigma)$. By the covering property, $\text{var}(C_t) = \text{var}(B_t)$. Then $\text{var}(C\sigma) = \text{var}(C_t\sigma)$. Then we have $\text{var}(C\sigma) = \text{var}(C_i\sigma)$ for all i such that $1 \leq i \leq m$. Since C is a flat clause, compound terms in the resolvent C' are inherited from $D_1, \ldots, D_m$. Let $\mathbb{G}$ be a loose guard and t a compound term in a C_i of $C_1, \ldots, C_m$. By **Definition 20**, $\text{var}(t) = \text{var}(\mathbb{G}) = \text{var}(C)$. Then $\text{var}(t\sigma) = \text{var}(\mathbb{G}\sigma) = \text{var}(C\sigma) = \text{var}(C_i\sigma)$ for all i such that $1 \leq i \leq m$. By *4.* in **Lemma 6.5**, $\mathbb{G}\sigma$ is flat and $t\sigma$ is a non-nested compound term. Hence, C' is simple, covering and contains a loose guard $\mathbb{G}\sigma$, hence, C' is an LG clause. □

Lemmas 6.7–6.8 prove that applying the **Fact** and **P-Res** rules (endowed with the **T-Ref^LGQ** refinement) to LG clauses derive LG clauses. This proves that the derived LG clauses are of bounded depth, as LG clauses are simple. Let us now investigate the width of derived LG clauses. Recall that by the width of a clause, we mean the number of distinct variables in that clause.

Lemma 6.9. *In applications of the* ***T-Inf^LGQ*** *system to LG clauses, the derived LG clause is no wider than at least one of its premises.*

Proof. By adapting 'guards' to 'loose guards' in the proof of **Lemma 4.17**. □

Now we give the first main result of this section.

Theorem 6.4. *The* ***T-Inf^LGQ*** *system decides satisfiability of the LG clausal class.*

Proof. Suppose $(\mathtt{C}, \mathtt{F}, \mathtt{P})$ is a finite set of signature for the given LG clauses. By **Lemmas 6.7–6.8**, applying the **T-Inf^LGQ** system to LG clauses derives the LG clauses with bounded depth. By **Lemma 6.9**, the derived LG clauses are of bounded width. These derived LG clauses only use symbols in $(\mathtt{C}, \mathtt{F}, \mathtt{P})$, as no symbols are introduced in the derivation. □

To properly end this section, we give a sample derivation to show how the **T-Inf$^{\mathsf{LGQ}}$** system decide an unsatisfiable set of LG clauses.

Consider an unsatisfiable set N of LG clauses $C_1, \ldots, C_9$:

$$C_1 = \neg A_1(x,y) \lor \neg A_2(y,z) \lor \neg A_3(z,x) \lor B(x,y,b),$$
$$C_2 = A_3(x, f(x)) \lor \neg G_3(x), \qquad C_3 = A_2(f(x), f(x)) \lor \neg G_2(x),$$
$$C_4 = A_1(f(x), x) \lor D(g(x)) \lor \neg G_1(x), \qquad C_5 = \neg B(x,y,b),$$
$$C_6 = \neg D(x), \quad C_7 = G_1(f(a)), \quad C_8 = G_3(f(a)), \quad C_9 = G_2(a).$$

Suppose the precedence on which $>_{lpo}$ is based is $f > g > a > b > B > A_1 > A_2 > A_3 > D > G_1 > G_2 > G_3$. Recall that by $\boxed{L}$ and L^* we mean the literal L is selected and L is the (strictly) maximal literal, respectively. Then by restrictions of the **T-Ref$^{\mathsf{LGQ}}$** refinement, $C_1, \ldots, C_9$ are presented as:

$$C_1 = \neg A_1(x,y) \lor \neg A_2(y,z) \lor \neg A_3(z,x) \lor B(x,y,b),$$
$$C_2 = A_3(x, f(x))^* \lor \neg G_3(x), \qquad C_3 = A_2(f(x), f(x))^* \lor \neg G_2(x),$$
$$C_4 = A_1(f(x), x)^* \lor D(g(x)) \lor \neg G_1(x), \qquad C_5 = \boxed{\neg B(x,y,b)},$$
$$C_6 = \boxed{\neg D(x)}, \quad C_7 = G_1(f(a))^*, \quad C_8 = G_3(f(a))^*, \quad C_9 = G_2(a)^*.$$

One can use any clause to start the derivation, w.l.o.g. we start with C_1. For each newly derived clause, **Algorithms** 14 is immediately applied to determine the (**PRes**) eligible literals of it.

1. By **Algorithms** 14 and the fact that C_1 is a flat LG clause, the P-Res function is used to C_1. By **Algorithms** 2, all negative literals in C_1 are selected to check if the **Res** rule is applicable to C_1.

2. As an **Res** inference step is applicable to C_2, C_3, C_4 (as the side premises) and C_1 (as the main premise), CompT(C_2, C_3, C_4, C_1) computes an mgu

$$\sigma' = \{x \mapsto f(f(x')), y \mapsto f(x'), z \mapsto f(x')\}$$

for variables of C_1. Hence in C_1, x is the only top variable and $\neg A_1(x,y)$ and $\neg A_3(z,x)$ are the **P-Res** eligible literals.

3. The top-variable resolution inference is applied to C_2, C_4 and C_1 with an

mgu $\sigma = \{x \mapsto f(x'), y \mapsto x', z \mapsto x'\}$, deriving

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

where x' is renamed as x. No resolution step can be performed on C_3 and C_{10} as they do not have complementary eligible literals, but an inference can be performed between C_5 and C_{10}.

4. Applying (the binary form of) the **P-Res** rule to C_5 and C_{10} derives

$$C_{11} = \neg A_2(x, x) \vee D(g(x))^* \vee \neg G_1(x) \vee \neg G_3(x).$$

5. Applying (the binary form of) the **P-Res** rule to C_6 and C_{11} derives

$$C_{12} = \neg A_2(x, x) \vee \neg G_1(x) \vee \neg G_3(x).$$

6. Since C_{12} is a flat LG clause, we apply the P-Res function to it. Due to the presence of C_3, C_7, C_8 and C_{12} satisfy conditions of the top-variable resolution rule, CompT(C_3, C_7, C_8, C_{12}) finds that x is the only top variable in C_{12}, using an mgu $\sigma' = \{x \mapsto f(a)\}$. Then all literals in C_{12} are selected. Applying the top-variable resolution rule to C_3, C_7, C_8 (as the side premises) and C_{12} (as the main premise) derives $C_{13} = \boxed{\neg G_2(x)}$.

7. Applying (the binary form of) the **P-Res** rule to C_9 and C_{13} derives $\perp$.

Given an LG clausal set, resolution refinement and the **P-Res** rule allow inferences building a model or deriving a contradictory without producing unnecessary conclusions. Using the **T-Ref**$^{\mathsf{LGQ}}$ refinement, fewer inferences are computed to derive a contradiction. For instance, in the previous example inferences between C_2 and C_8 and between C_3 and C_9 are prevented since these pairs do not contain complementary eligible literals. These unnecessary inferences would be computed if there is no refinement guiding resolution.

Note that in the previous example, one can also select the literal $\neg A_2(x, x)$ in C_{12} as C_{12} is a flat guarded clause. The fact that in the **T-Ref**$^{\mathsf{LGQ}}$ refinement, one can use the SelectG function to flat guarded clauses is discussed in **Section 6.2**.

Handing query clauses (in the presence of LG clauses)

In this section, we consider deciding satisfiability of the whole of LGQ clausal class. In particular we handle query clauses in the presence of LG clauses. By **Lemma 4.23**, the **Q-Sep** procedure separates query clauses into Horn guarded clauses (HG clauses) and indecomposable chained-only query clauses (CO clauses), therefore we focus on investigating the inferences performed on indecomposable CO clauses and LG clauses.

Recall that in **Section 4.5**, the **T-Trans** rule transforms the top-variable resolvents of an indecomposable CO clause and guarded clauses to GQ clauses. In this section, we abusively reuse the notion **T-Trans** to denote the rule that handles the top-variable resolvents of an indecomposable CO clause and LG clauses. This new **T-Trans** rule is the same as the **T-Trans** rule in **Section 4.5**, except that in (the **P-Res** rule of) this new **T-Trans** rule, the side premises are not guarded clauses, but LG clauses.

The **T-Trans** rule transforms the resolvents of an indecomposable CO clause and LG clauses to LGQ clauses. This result is formally reported as follows.

Lemma 6.10. *Let R be the resolvent of applying the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}}$ refinement) to an indecomposable CO clause Q and a set N of LG clauses. Then, the following conditions hold.*

1. *Applying the **T-Trans** rule to R replaces it by a set N' of LG clauses and a query clause Q'.*
2. *Applying the **Q-Sep** procedure to Q' separates it into a set N'' of HG clauses and an indecomposable CO clause Q''.*
3. *The top-resolvent R is satisfiable if and only if the LGQ clausal set $N' \cup N'' \cup Q''$ is satisfiable.*
4. *For each clause C' in $N' \cup N''$, there exists a clause C in N such that C' is no wider than C, and Q'' is less wide than Q.*

Proof. By **Lemma 6.5** and by adapting the notion of 'guard' to that of 'loose guard' in the proof of **Lemma 4.25**. □

We use the notation **Q-CO**$^{\mathsf{LGQ}}$ to denote the procedure for handling CO clauses in the presence of LG clauses, given as follows.

1. Apply the top-variable resolution rule to an indecomposable CO clause and LG clauses, deriving the top-variable resolvent R.

2. Apply the **T-Trans** rule to R, deriving a query clause Q and LG clauses.

3. Apply the **Q-Sep** procedure to Q, producing HG clauses and an indecomposable CO clause.

The result of handling indecomposable CO clauses (in the presence of LG clauses) is formally stated as:

Lemma 6.11. *The conclusions of applying the* **Q-CO$^{\mathsf{LGQ}}$** *procedure to an indecomposable* CO *clause Q and a set N of* LG *clauses satisfy the following conditions.*

1. *The conclusions are an indecomposable* CO *clause Q' and a set N' of* LG *clauses.*
2. *The clausal sets $Q' \cup N'$ and $Q \cup N$ are equisatisfiable.*
3. *For each clause C' in N', there exists a clause C in N such that C' is no wider than C, and Q' is less wide than Q.*

Proof. By **Lemmas** 4.23 and 6.10, *1.* and *3.* hold. By **Lemma** 3.4 and the fact that any form of structural transformation rule preserves satisfiability, *2.* hold. □

6.4 Decision procedures of querying in LGF and/or CGF

A BCQ answering procedure for LGF and CGF

In this section, we present the saturation-based decision procedure for answering BCQs for LGF and/or CGF. Like the saturation-based BCQ answering procedure for GF (see **Algorithm** 5), the procedure of querying LGF and/or CGF is also devised in line with the give-clause algorithms in [Wei01, MW97].

We start with introducing the BCQ answering procedure for LGF and CGF, and we use the notation **Q-Ans$^{\mathsf{CGF}}$** to denote this procedure. The **Q-Ans$^{\mathsf{CGF}}$** procedure consists of the same functions as the **Q-Ans$^{\mathsf{GF}}$** procedure, except the PreProcessCGF function and the fact that the input clauses are the LG clauses, rather than the guarded clauses. See **Algorithm** 15 on the next page. We refer readers to **Section** 4.6 for the detailed descriptions of the functions and the processes in **Algorithm** 15.

Algorithm 15: The BCQ answering procedure for LGF and CGF

Input: A union q of BCQs, sets Σ_1 and Σ_2 of formulas in LGF and CGF, respectively

Output: 'Yes' or 'No'

workedOff $\leftarrow \emptyset$
usable $\leftarrow$ PreProcessCGF(Σ_1, Σ_2, q)
while usable $= \emptyset$ *and* $\bot \notin$ usable **do**
    given $\leftarrow$ Pick(usable)
    workedOff $\leftarrow$ workedOff $\cup$ given
    if given *is an indecomposable CO clause* **then**
        tResolvent $\leftarrow$ P-Res(workedOff, given)
        G, Q $\leftarrow$ T-Trans(tResolvent)
        CO, HG $\leftarrow$ Sep(Q)
        new $\leftarrow$ G $\cup$ CO $\cup$ HG
    else
        new $\leftarrow$ P-Res(workedOff, given) $\cup$ Fact(given)
    new $\leftarrow$ Red(new, new)
    new $\leftarrow$ Red(Red(new, workedOff), usable)
    workedOff $\leftarrow$ Red(workedOff, new)
    usable $\leftarrow$ Red(usable, new) $\cup$ new
Print(usable)

Next, **Algorithm** 16 describes the PreProcessCGF(Σ_1, Σ_2, q) function, which pre-processes a union q of BCQs and sets Σ_1 and Σ_2 of formulas LGF and CGF, respectively, transforming these formulas to indecomposable CO clauses and LG clauses. In **Algorithm** 16, the notations LG_1 and LG_2 are used to denote the LG clausal sets that are obtained from LGF and CGF, respectively.

Comparing to **Algorithm** 6 that handles GF and BCQs, **Algorithm** 16 contains the following novel functions.

1. TransGF(Σ, q) applies the $\mathbf{Trans}^{\mathsf{GF}}$ process to a set Σ of formulas in LGF and a union q of BCQs, returning LG clauses and query clauses.

2. TransCGF(Σ, q) applies the $\mathbf{Trans}^{\mathsf{CGF}}$ process to a set Σ of formulas in CGF and a union q of BCQs, returning LG clauses and query clauses.

Like the **Q-Ans**$^{\mathsf{GF}}$ procedure, the **Q-Ans**$^{\mathsf{CGF}}$ procedure reuses predicate symbols in the derivation. By reusing, we mean that in the **Q-Ans**$^{\mathsf{CGF}}$ procedure, if a predicate symbol P is used to represent a LGQ clause C at a derivation stage, then, in any further derivation step whenever a predicate symbol is needed for C, we use the symbol P again.

Algorithm 16: The PreProcessCGF function

Input: A union q of BCQs, sets Σ_1 and Σ_2 of formulas in LGF and CGF, respectively
Output: A set of indecomposable CO and LG clauses

Function PreProcessCGF(Σ_1, Σ_2, q):
  usable $\leftarrow \emptyset$
  LG$_1$, Q $\leftarrow$ TransGF(Σ_1, q)
  LG$_2$, Q $\leftarrow$ TransCGF(Σ_2, q)
  foreach *clause* Q *in* Q **do**
    CO, HG $\leftarrow$ Sep(Q)
    usable $\leftarrow$ usable $\cup$ CO $\cup$ HG
  usable $\leftarrow$ usable $\cup$ LG$_1$ $\cup$ LG$_2$
  usable $\leftarrow$ Red(usable, usable)
  return usable

Lemma 6.12. *In the application of the* **Q-Ans**$^{\mathsf{CGF}}$ *procedure to the* BCQ *answering problem for* LGF *and/or* CGF*, only finitely many predicate symbols are introduced.*

Proof. By adapting the notion of 'guard' to that of 'loose guard' in the proof of **Lemma** 4.27. □

Finally, we give a positive answer to **Problem** 6.

Theorem 6.5. *The* **Q-Ans**$^{\mathsf{CGF}}$ *procedure is a decision procedure for answering* BCQs *for* LGF *and/or* CGF*.*

Proof. By **Theorems** 6.1–6.2, the problems of answering BCQs for LGF and/or CGF are reduced to that of deciding satisfiability of the LGQ clausal class. By **Lemma** 4.19, **Theorem** 6.3 and the fact that the **Q-Ans**$^{\mathsf{CGF}}$ procedure is based on the **T-Inf**$^{\mathsf{LGQ}}$ system, the **Q-Ans**$^{\mathsf{CGF}}$ procedure is sound and refutational complete for general first-order clausal logic (if only finitely many predicate symbols are introduced in the derivation).

By **Lemma** 4.23, **Lemma** 6.11 and **Theorem** 6.4, applying the **Q-Ans**$^{\mathsf{LGF}}$ procedure to LGQ clauses guarantees producing LGQ clauses of bounded depth and bounded width. By **Lemma** 6.12, only finitely many new predicate symbols are introduced. Thus the **Q-Ans**$^{\mathsf{GF}}$ procedure guarantees termination. The **Q-Ans**$^{\mathsf{GF}}$ procedure is sound, refutationally complete for first-order clausal logic and guarantees termination for the LGQ clausal class, hence it is a decision procedure for answering BCQs for LGF and/or CGF. ◻

A saturation-based BCQ rewriting procedure for LGF and CGF

Deciding satisfiability of the LGQ$^-$ clausal class

In this section we give a more refined clausal form of LGF and CGF, namely the *aligned loosely guarded clauses*, and then formally prove that the **Q-Ans**$^{\mathsf{CGF}}$ procedure decides satisfiability of the *aligned loosely guarded clausal class* and query clauses.

Recall the saturation-based rewriting problem for BCQs with LGF and/or CGF.

Problem 7. *Given a set* Σ *of formulas in LGF and/or CGF, a set* D *of ground atoms and a union* q *of BCQs, does there exist a (function-free) first-order formula (with equality)* Σ_q *that is the negated back-translation of the saturated clausal set of* $\Sigma \cup \{\neg q\}$ *such that* $\Sigma \cup \mathrm{D} \models q$ *if and only if* $\mathrm{D} \models \Sigma_q$*?*

We define a more specific clausal form of LGF and CGF as follows.

Definition 21. *An* aligned loosely guarded clause (LG$^-$ clause) *is strongly compatible and an LG clause.*

We use the notation LGQ$^-$ to denote the class of LG$^-$ clauses and query clauses. The LG$^-$ clausal class is a strict subset of that of the LG clausal class.

Lemma 6.13. *i) Applying the* ***Trans***$^{\mathsf{GF}}$ *process to a loosely guarded formula transforms it into a set of LG*$^-$ *clauses, and ii) applying the* ***Trans***$^{\mathsf{CGF}}$ *process to a clique guarded formula transforms it into a set of LG*$^-$ *clauses.*

Proof. By **Lemma** 6.1 and **Lemma** 6.2, the **Trans**$^{\mathsf{GF}}$ process and the **Trans**$^{\mathsf{CGF}}$ process transform, respectively, loosely guarded formulas and clique guarded formulas to either flat LG clauses or non-ground compound-term LG clauses. Hence, all ground clauses in the LG clausal class are flat, which satisfies *1.* in **Definition 21**. In this proof, we focus on proving that all compound terms of the non-ground compound-term LG clauses are compatible.

Compounds terms in LG clauses are derived by Skolemising existential quantified variables. By the proofs of **Lemmas 6.1**–**6.2**, compound terms are obtained from Skolemising the definition formula

$$F = \forall \overline{xy}(\neg P(\overline{x}) \lor \neg G_1 \lor \ldots \lor \neg G_n \lor \phi(\overline{y}))$$

where i) $\phi(\overline{y})$ is a formula of atoms and existentially quantified formulas that are connected by Boolean connectives, and ii) each pair of distinct variables in $\overline{x} \cup \overline{y}$ co-occurs in a literal of $\neg P(\overline{x}) \lor \neg G_1(\ldots) \lor \ldots \lor \neg G_n(\ldots)$. Note that $\phi(\overline{y})$ contains no universal quantifications. Then for all existential quantified variables in $\phi(\overline{y})$, they are Skolemised into the Skolem compound terms that are with the same argument list $\overline{xy}$. Hence all compound terms in non-ground compound-term LG clauses are compatible. □

Next, we prove that the **Q-Ans**$^{\mathsf{CGF}}$ procedure decides the LGQ$^-$ clausal class. By the covering property of the LGQ$^-$ clauses, an *a priori checking* is used in the application of the **Q-Ans**$^{\mathsf{CGF}}$ procedure for the LGQ$^-$ clausal class.

We start with considering the application of the **Fact** rule to LGQ$^-$ clauses.

Lemma 6.14. *Applying the **Fact** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}}$ refinement) to LGQ$^-$ clauses derives LGQ$^-$ clauses.*

Proof. By **Algorithm 14**, the **Fact** rule is only applicable to LG$^-$ clauses. By adapting 'guards' to 'loose guards' in the proof of **Lemma 5.3**, applying the **Fact** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}}$ refinement) to LG$^-$ clauses derives LG$^-$ clauses. □

Next, we consider applying the **P-Res** rule to LGQ$^-$ clauses.

Lemma 6.15. *Applying the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}}$ refinement) to LGQ$^-$ clauses derives LGQ$^-$ clauses.*

Proof. In this proof, we discuss the inference **I** when the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}}$ refinement) is applied to a flat LG$^-$ clause (as the main premise) and compound-term LG$^-$ clauses (as the side premises). By

Lemma 6.8, the conclusions of **I** are LG clauses. Hence we focus on proving that all compound terms in these derived LG clauses are strongly compatible. For the rest of cases of applying the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}}$ refinement) to LGQ$^-$ clauses, their results can be easily obtained by adapting the proofs in **Lemmas** 5.4 and 5.5.

Assume the top-variable resolution rule is applied to compound-term LG$^-$ clauses $C_1 = B_1 \vee D_1, \ldots, C_n = B_n \vee D_n$ as the side premises, and a flat LG$^-$ clause $C = \neg A_1 \vee \ldots \vee \neg A_m \vee \ldots \vee \neg A_n \vee D$ as the main premise, deriving the resolvent $R = (D_1 \vee \ldots \vee D_m \vee \neg A_{m+1} \vee \ldots \vee \neg A_n \vee D)\sigma$ where σ is an mgu such that $\sigma = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_m \doteq B_m)$. By the fact that C is flat, compound terms in R come from $D_1, \ldots, D_m$. W.l.o.g. assume that s and u are compound terms in D_1 and t is a compound term in D_1. To show all compound terms in R are compatible, one needs to show that $s\sigma$, $u\sigma$ and $t\sigma$ are compatible. By *1.* of **Lemma** 5.2, $s\sigma$ and $u\sigma$ are compatible. Now we prove that $s\sigma$ and $t\sigma$ are compatible. By **Algorithm** 14, B_1 and B_2 are compound-term literals. Then suppose s' and t' are compound terms in B_1 and B_2, respectively. By **Lemma** 6.5, s' and t' pair top variables. W.l.o.g. suppose s' and t' pair top-variables x_1 and x_2 in A_1 and A_2, respectively. By *2.* of **Definition** 21, suppose x_1 and x_2 co-occur in the literal $\neg A_i$ of $\neg A_1, \ldots, \neg A_m$. By **Lemma** 6.5, x_1 and x_2 pair compound terms in B_i. Suppose x_1 and x_2 pair s'' and t'' in B_i, respectively. As all compound terms in C_i are compatible, s'' and t'' are compatible. By *1.* of **Lemma** 5.2, $s''\sigma$ and $t''\sigma$ are compatible, hence $x_1\sigma$ and $x_2\sigma$ are compatible. By the facts that x_1 pairs s' and x_2 pairs t', $s'\sigma$ and $t'\sigma$ are compatible. By *3.* of **Lemma** 5.2 and the facts that s' and t' are compatible with s and t, respectively, $s\sigma$ and $t\sigma$ are compatible. By the fact that $s\sigma$, $t\sigma$ and $u\sigma$ are compatible, all compound terms in R are compatible. □

There are finitely many new predicate symbols that are introduced in applying the **Q-Ans**$^{\mathsf{CGF}}$ procedure to the LGQ$^-$ clausal class. This result immediately follows from **Lemma** 6.12, since the class of LGQ$^-$ clauses is a strict subset of that of LGQ clauses.

Theorem 6.6. *The* ***Q-Ans***$^{\mathsf{CGF}}$ *procedure decides satisfiability of the* LGQ$^-$ *clausal class.*

Proof. By **Lemma** 6.12 and **Lemmas** 6.14–6.15. □

Back-translating LGQ$^-$ clausal sets to first-order formulas

In this section, we first give the procedure that back-translates LGQ$^-$ clausal sets into a first-order formula with equality, but without Skolem symbols, and we then present our saturation-based rewriting procedure for BCQs with LGF and/or CGF, with a complete example.

Recall that given a clausal set N, N can be back-translated to a first-order formula if N can be transformed into a unique, normal, globally linear and globally compatible clausal set. This transformation requires one to align argument lists of compound terms of clauses in N. Since in LGQ$^-$ clauses, loose guards contain no compound terms, one can use the back-translation procedure for the GQ$^-$ clausal class to back-translate LGQ$^-$ clausal sets. We abusively use notations **Q-Abs**, **Q-Rena** and **Q-Unsko** (for transforming GQ$^-$ clausal sets) to back-translate LGQ$^-$ clausal sets to a first-order formula, with the restriction that in these procedures, the conditions are changed from 'guard' to 'loose guard'.

Lemma 6.16. *Suppose N is an LGQ$^-$ clausal set. Then, the following condition hold.*

1. *N is a locally linear and locally compatible clausal set.*
2. *Applying the **Q-Abs** procedure to N transforms N to a normal, unique, locally linear and locally compatible clausal set N_1.*
3. *Applying the **Q-Rena** procedure to N_1 transforms N_1 to a normal, unique, globally linear and globally compatible clausal set N_2.*
4. *Applying the **Q-Unsko** procedure to N_2 transform it to a first-order formula without Skolem symbol, but with equality.*

Proof. We adapt the notion of 'guard' to that of 'loose guard' in the proofs of the following lemmas. By **Lemma** 3.2, *1.* holds. By **Lemmas** 5.6–5.7, *2.* holds. By **Lemma** 5.10, *3.* holds. By **Lemma** 5.12, *4.* holds. □

We use the notation **Q-Rew$^{\mathsf{CGF}}$** to denote our saturation-based rewriting procedure for BCQs with LGF and/or CGF. Given a union q of BCQs, a set Σ of formulas in LGF and/or CGF and a set D of ground atoms, to compute a first-order formula the negated back-translation of $\Sigma \cup \{\neg q\}$, the **Q-Rew$^{\mathsf{CGF}}$** procedure uses the following steps.

1. Apply the **Q-Ans$^{\mathsf{CGF}}$** procedure to $\Sigma \cup \{\neg q\}$, producing a set N of LGQ$^-$ clauses.

2. Apply the **Q-Abs** procedure to N, obtaining a normal, unique, strongly compatible clausal set N_1.

3. Apply the **Q-Rena** procedure to N_1, obtaining a normal, unique, globally linear and globally compatible clausal set N_2.

4. Apply the **Q-Unsko** procedure to N_2, obtaining a first-order formula F.

5. Negate F, obtaining Σ_q.

By **Theorem** 5.4, we give a positive answer to **Problem** 7. This is also the second main contribution of this chapter.

Theorem 6.7. *Suppose Σ is a set of formulas in LGF and/or CGF, D is a set of ground atoms and q is a union of BCQs. The* **Q-Rew$^{\mathsf{CGF}}$** *procedure is the decision procedure that back-translates, and then negates the saturated clausal set of $\Sigma \cup \{\neg q\}$ to a (function-free) first-order formula with equality Σ_q such that $\Sigma \cup \mathrm{D} \models q$ if and only if $\mathrm{D} \models \Sigma_q$.*

Proof. By **Lemma** 6.16. □

To end this chapter, we use the following rewriting problem as an example to show how the **Q-Rew$^{\mathsf{CGF}}$** procedure is performed. Given a union q of BCQs, a set Σ of formulas in CGF, a set D of dataset, the **Q-Ans$^{\mathsf{CGF}}$** procedure computes the saturation of $\Sigma \cup \{\neg q\}$ as

$$N = \left\{ \begin{array}{l} \neg G_1(x_1, a) \vee A_1(f(x_1, a), x_1) \vee A_2(g(x_1, a), x_1), \\ \neg G_2(x_2, x_3) \vee A_3(f(x_2, x_3), x_2) \vee A_4(g(x_2, x_3), x_2), \\ \neg G_3(b, x_4) \vee A_5(g(b, x_4), b) \\ \neg G_4(x_5, c, c) \vee A_6(h(c, c, x_5)) \\ \neg B_1(x_8, x_6) \vee \neg B_2(x_6, x_7) \vee \neg B_3(x_7, x_8) \end{array} \right\}$$

where a and c are non-Skolem constants and b is a Skolem constant. Now we aim to back-translate N to a first-order formula Σ_q such that $\mathrm{D} \models \Sigma_q$ if and only if $\Sigma \cup \mathrm{D} \models q$. The **Q-Rew$^{\mathsf{CGF}}$** procedure back-translates N to Σ_q as follows.

In the first step, the **Q-Abs** procedure is applied to N, given as follows.

1. For each clause in N, recursively applying the **ConAbs** rule to it, obtaining

$$N_1 = \left\{ \begin{array}{l} \neg G_1(x_1, y_1) \vee A_1(f(x_1, y_1), x_1) \vee A_2(g(x_1, y_1), x_1) \vee y_1 \not\approx a, \\ \neg G_2(x_2, x_3) \vee A_3(f(x_2, x_3), x_2) \vee A_4(g(x_2, x_3), x_2), \\ \neg G_3(y_2, x_4) \vee A_5(g(y_2, x_4), y_2) \vee y_2 \not\approx b, \\ \neg G_4(x_5, y_3, y_3) \vee A_6(h(y_3, y_3, x_5)) \vee y_3 \not\approx c \\ \neg B_1(x_8, x_6) \vee \neg B_2(x_6, x_7) \vee \neg B_3(x_7, x_8) \end{array} \right\}.$$

2. For each clause in N_1, recursively apply the **VarAbs** rule to it, obtaining

$$N_2 = \left\{ \begin{array}{l} \neg G_1(x_1, y_1) \vee A_1(f(x_1, y_1), x_1) \vee A_2(g(x_1, y_1), x_1) \vee y_1 \not\approx a, \\ \neg G_2(x_2, x_3) \vee A_3(f(x_2, x_3), x_2) \vee A_4(g(x_2, x_3), x_2), \\ \neg G_3(y_2, x_4) \vee A_5(g(y_2, x_4), y_2) \vee y_2 \not\approx b, \\ \neg G_4(x_5, y_3, y_4) \vee A_6(h(y_3, y_4, x_5)) \vee y_3 \not\approx c \vee y_4 \not\approx y_3 \\ \neg B_1(x_8, x_6) \vee \neg B_2(x_6, x_7) \vee \neg B_3(x_7, x_8) \end{array} \right\}.$$

In the second step, the **Q-Rena** procedure is applied to N_2.

1. Partition N_2 into closed clausal sets

$$N_2' = \left\{ \begin{array}{l} \neg G_1(x_1, y_1) \vee A_1(f(x_1, y_1), x_1) \vee A_2(g(x_1, y_1), x_1) \vee y_1 \not\approx a, \\ \neg G_2(x_2, x_3) \vee A_3(f(x_2, x_3), x_2) \vee A_4(g(x_2, x_3), x_2), \\ \neg G_3(y_2, x_4) \vee A_5(g(y_2, x_4), b) \vee y_2 \not\approx b \end{array} \right\},$$

$$N_2'' = \left\{ \neg G_4(x_5, y_3, y_4) \vee A_6(h(y_3, y_4, x_5)) \vee y_3 \not\approx c \vee y_4 \not\approx y_3 \right\},$$

and $N_2''' = \{\neg B_1(x_8, x_6) \vee \neg B_2(x_6, x_7) \vee \neg B_3(x_7, x_8)\}$.

2. Since N_2' and N_2'' are inter-connected clausal sets and N_2''' is a compound-term-free clausal set, the **VarRe** rule is only applied to N_2' and N_2''. As compound terms in N_2' are binary, a new variable sequence x, y (with respect to N_2') is used to rename all variables in N_2', transforming N_2' into

$$N_3' = \left\{ \begin{array}{l} \neg G_1(x, y) \vee A_1(f(x, y), x) \vee A_2(g(x, y), x) \vee y \not\approx a, \\ \neg G_2(x, y) \vee A_3(f(x, y), x) \vee A_4(g(x, y), x), \\ \neg G_3(x, y) \vee A_5(g(x, y), x) \vee x \not\approx b \end{array} \right\}.$$

A new variable sequence x_1, y_1, z_1 (with respect to N_2'') is used to rename all variables in N_2'', transforming N_2'' into

$$N_3'' = \left\{ \neg G_4(x_1, y_1, z_1) \vee A_6(h(y_1, z_1, x_1)) \vee y_1 \not\approx c \vee z_1 \not\approx y_1 \right\}.$$

3. Eventually, from N_2, we obtain the clausal set $N_3' \cup N_3'' \cup N_2'''$.

In the third step, the **Q-Unsko** procedure is used to unskolemise $N_3' \cup N_3'' \cup N_2'''$.

1. As N_3' and N_3'' are inter-connected clausal sets, the **UnskoOne** rule is applied to these clausal sets. Applying the **UnskoOne** rule to N_3' transforms it into

$$F_1 = \exists z' \forall xy \exists x'y' \begin{bmatrix} (\neg G_1(x, y) \vee A_1(x', x) \vee A_2(y', x) \vee y \not\approx a) & \wedge \\ (\neg G_2(x, y) \vee A_3(x', x) \vee A_4(y', x)) & \wedge \\ (\neg G_3(x, y) \vee A_5(y', x) \vee x \not\approx z') & \end{bmatrix},$$

and applying **UnskoOne** rule to N_3'' transforms it into

$$F_2 = \forall x_1 y_1 z_1 \exists x_1' \left[\neg G_4(x_1, y_1, z_1) \vee A_6(x_1') \vee A_7(x_1') \vee y_1 \not\approx c \vee z_1 \not\approx y_1 \right].$$

2. As N_2''' is a compound-term-free clause set, applying the **UnskoTwo** rule to N_2''' unskolemise it into

$$F_3 = \forall x_6 x_7 x_8 \left[\neg B_1(x_8, x_6) \vee \neg B_2(x_6, x_7) \vee \neg B_3(x_7, x_8) \right].$$

3. Finally N is back-translated into a first-order formula $F = F_1 \wedge F_2 \wedge F_3$.

In the last step, F is negated to obtain Σ_q, given as follows.

$$\forall z' \exists xy \forall x'y' \begin{bmatrix} (G_1(x, y) \wedge \neg A_1(x', x) \wedge \neg A_2(y', x) \wedge y \approx a) & \vee \\ (G_2(x, y) \wedge \neg A_3(x', x) \wedge \neg A_4(y', x)) & \vee \\ (G_3(x, y) \wedge \neg A_5(y', x) \wedge x \approx z') & \end{bmatrix} \bigvee$$
$$\exists x_1 y_1 z_1 \forall x_1' \left[G_4(x_1, y_1, z_1) \wedge \neg A_6(x_1') \wedge \neg A_7(x_1') \wedge y_1 \approx c \wedge z_1 \approx y_1 \right] \vee$$
$$\exists x_6 x_7 x_8 [B_1(x_8, x_6) \wedge B_2(x_6, x_7) \wedge B_3(x_7, x_8)]$$

Chapter 7

Querying for GNF and CGNF

In this chapter we focus on the querying in the guarded negation fragments.

Problem 8. *Given a set Σ of formulas* GNF *and/or* CGNF *and a union q of* BCQs, *does there exist a practical decision procedure that decides $\Sigma \models q$?*

As for the saturation-based BCQ rewriting problem, we consider a more challenging problem, that is the back-translation of the saturation to a (clique) guarded negation formula.

Problem 9. *Given a set Σ of (clique) guarded negation formulas, a set D of ground atoms and a union q of* BCQs, *does there exist a (clique) guarded negation formula Σ_q that is the negated back-translation of the saturated clausal set of $\Sigma \cup \{\neg q\}$ such that $\Sigma \cup \mathrm{D} \models q$ if and only if $\mathrm{D} \models \Sigma_q$?*

Note that in this chapter we consider BCQ as BCQ with equality as equality is allowed in GNF and CGNF.

7.1 Clausifications for GNF and CGNF

Transforming GNF to the $\mathsf{GQ}_{\approx}$ clausal class

Recall the definition of GNF from **Section 2.1**.

Definition 5. *The* guarded negation fragment (GNF) *is a fragment of* $\mathsf{FOL}_{\approx}$ *without functional symbols, inductively defined as follows:*

1. *$\top$ and $\bot$ belong to* GNF.

2. *If A is an atom, then A belongs to* GNF*.*
3. *If A and B are atoms, then $A \vee B$ and $A \wedge B$ belong to* GNF*.*
4. *If F belongs to* GNF*, then $\exists \overline{x} F$ belongs to* GNF*.*
5. *Let F be a guarded negation formula and G an atom. Then $G \wedge \neg F$ belongs to* GNF *if all free variables of F belong to the variables of G.*

We use the notation **Trans**$^{\mathsf{GNF}}$ to denote our customised structural transformation for guarded negation formulas and a union of BCQs with equality. In the first step, the **Trans**$^{\mathsf{GNF}}$ process negates the given union of BCQs with equality, obtaining a set of $Q_{\approx}$ *clauses*. In the second step of the **Trans**$^{\mathsf{GNF}}$ process, guarded negation formula are transformed into clauses. We use the guarded negation formula

$$F = E(x, y) \wedge \neg\exists uvw(E(x, u) \wedge E(u, v) \wedge E(v, w) \wedge E(w, y))$$

as an example to show how the **Trans**$^{\mathsf{GNF}}$ process is performed.

1. Add existential quantifiers for all free variables in F, obtaining

$$F_1 = \exists xy(E(x, y) \wedge \neg\exists uvw(E(x, u) \wedge E(u, v) \wedge E(v, w) \wedge E(w, y))).$$

2. Apply the **Trans** rules to F_1, introducing fresh predicate symbols P (and respective literals $P(\ldots)$) for all occurrences of the guarded negation pattern $G \wedge \neg F'$ that occur in F_2, obtaining

$$F_2 = \begin{bmatrix} \exists xy(& P(x, y) &)\wedge \\ \forall xy(& P(x, y) \rightarrow (E(x, y) \wedge \neg\exists uvw(& \\ & E(x, u) \wedge E(u, v) \wedge E(v, w) \wedge E(w, y))) &) \end{bmatrix}.$$

   We say that

   - $\exists xyP(x, y)$ is the *replacing formula of* F_1, and
   - $\forall xy(P(x, y) \rightarrow (E(x, y) \wedge \neg\exists uvw(E(x, u) \wedge E(u, v) \wedge E(v, w) \wedge E(w, y))))$ is the *definition formula of* P.

3. Apply the **NNF** rules to F_2 to transform it to negation normal form,

obtaining

$$F_3 = \left[\begin{array}{lll} \exists xy(& P(x,y) &)\wedge \\ \forall xy(& \neg P(x,y) \vee (E(x,y) \wedge \forall uvw(& \\ & \neg E(x,u) \vee \neg E(u,v) \vee \neg E(v,w) \vee \neg E(w,y))) &) \end{array}\right].$$

4. Transform immediate subformulas (that are connected by conjunctions) of F_3 to prenex normal form and then apply the **Skolem** rule to these subformulas, eliminating their existential quantifications and existentially quantified variables. Then we obtain

$$F_4 = \left[\begin{array}{lll} & P(a,b) & \wedge \\ \forall xyuvw(& \neg P(x,y) \vee (E(x,y) \wedge (& \\ & \neg E(x,u) \vee \neg E(u,v) \vee \neg E(v,w) \vee \neg E(w,y))) &) \end{array}\right].$$

5. Apply the **CNF** rules to F_4 to transform it to conjunctive normal form, and then drop all universal quantifiers. Finally we obtain a set of clauses:

$$\left\{\begin{array}{l} P(a,b), \\ \neg P(x,y) \vee E(x,y), \\ \neg P(x,y) \vee \neg E(x,u) \vee \neg E(u,v) \vee \neg E(v,w) \vee \neg E(w,y). \end{array}\right\}.$$

Unlike the $\textbf{Trans}^{\textsf{GF}}$ and the $\textbf{Trans}^{\textsf{CGF}}$ processes, this $\textbf{Trans}^{\textsf{GNF}}$ process uses a more exhaustive structural transformation. In 2. of the $\textbf{Trans}^{\textsf{GNF}}$ process, we abstract all occurrences of the guarded negation pattern in F_1, and this step is applied before F_1 is transformed to negation normal form. The current formula renaming process gives us a better picture of the clausal forms of guarded negation formulas, even though the essential step in 2. is renaming the (implicit and explicit) universally quantified formulas, which are in the form of the guarded negation $G(\overline{y}) \wedge \neg\exists\overline{x}\psi(\overline{x},\overline{y})$. See details in the proofs of **Lemma 7.1**.

Definition 22. *A* guarded clause with equality ($\mathsf{G}_{\approx}$ clause) *C is a simple and covering clause that may contain equality, satisfying the following conditions:*

1. *C is either ground, or*
2. *C is a positive and single-variable clause, or*

3. *C contains a negative flat literal* $\neg G$ *satisfying* $\mathrm{var}(C) = \mathrm{var}(G)$.

Definition 23. *A* query clause with equality ($\mathsf{Q}_{\approx}$ clause) *is a flat and negative clause that may contain inequality literals.*

In *3.* of **Definition 24**, the literal $\neg G$ is called the *guard* of the clause C. We use the notation $\mathsf{GQ}_{\approx}$ to denote the class of $\mathsf{G}_{\approx}$ clauses and $\mathsf{Q}_{\approx}$ clauses.

The $\mathsf{G}_{\approx}$ clausal class strictly extends the guarded clausal class by allowing equality literals. Note that by simplifying the $\mathsf{G}_{\approx}$ clauses in which inequality literal are guards, one obtains flat and single variable clauses, as defined in *2.* of **Definition 22**. For example, the $\mathsf{G}_{\approx}$ clause $x \not\approx y \vee A(x, y) \vee B(x, x)$ is immediately simplified as $A(x, x) \vee B(x, x)$. This simplification step is achieved by the **E-Res** rule, which is discussed in **Lemma 7.8** from **Section 7.3**.

Note that in [GdN99, Definition 4.2], the clause C in *2.* of **Definition 22** is defined as 'a positive, non-functional, single-variable clause'. We relax this condition by defining C as a single-variable clause, since C may contain compound terms. For example, by the **NNF**, the **Skolem** and the **CNF** rules, the guarded negation formula $\neg\exists x(x \approx x \wedge \neg\exists y R(x, y))$ (or the guarded formula with equality $\forall x(x \approx x \rightarrow \exists y R(x, y)))$ is transformed into $R(x, f(x))$, which is not a positive, non-functional and single-variable clause. By the $\mathbf{Trans}^{\mathsf{GF}}$ and the $\mathbf{Trans}^{\mathsf{GNF}}$ processes, one also obtains $R(x, f(x))$ from $\forall x(x \approx x \rightarrow \exists y R(x, y))$ and $\neg\exists x(x \approx x \wedge \neg\exists y R(x, y))$, respectively.

Lemma 7.1. *The* $\mathbf{Trans}^{\mathsf{GNF}}$ *process transforms a guarded negation formula to a set of* $\mathsf{GQ}_{\approx}$ *clauses.*

Proof. Let F be a guarded negation formula. In this proof, we show that how the $\mathbf{Trans}^{\mathsf{GNF}}$ process transforms F to a $\mathsf{GQ}_{\approx}$ clausal set.

By *2.* of the $\mathbf{Trans}^{\mathsf{GNF}}$ process, we use predicate symbols P_1 and P_2 to abstract positive and negative occurrences of the guarded pattern in F, respectively. W.l.o.g. suppose F' is the replacing formula of F, $F_1 = \forall\overline{x}(P_1(\overline{x}) \rightarrow G(\overline{x}) \wedge \neg F')$ is the definition formula of P_1 and $F_2 = \forall\overline{x}(G(\overline{x}) \wedge \neg F' \rightarrow P_2(\overline{x}))$ is the definition formula of P_2. Now we prove that by *3.–5.* of the $\mathbf{Trans}^{\mathsf{GNF}}$ process, F', F_1 and F_2 are transformed to $\mathsf{GQ}_{\approx}$ clauses. We distinguish cases of F', F_1 and F_2 as follows.

i.: Consider F'. By the facts that F' is obtained by abstracting all guarded negation pattern from F and the universal quantifications in the F are only expressed by the guarded negation pattern $G(\overline{y}) \wedge \neg\exists\overline{x}\psi(\overline{x}, \overline{y})$, F' contains no universal quantifications. Hence F' is an existentially quantified sentence containing only flat and positive literals. Hence, by *4.* of the **Trans**$^{\mathsf{GNF}}$ process, F' is Skolemised to a (set of) flat and ground clause (if conjunctions occur in F'), which satisfies *1.* of the **Definition** 22. Hence, F' is a $\mathsf{G}_{\approx}$ clause.

ii.: Consider F_1. By *3.* of the **Trans**$^{\mathsf{GNF}}$ process, $\forall \overline{x}(P_1(\overline{x}) \rightarrow G(\overline{x}) \wedge \neg F')$ is transformed to $\neg P_1(\overline{x}) \vee G(\overline{x})$ and $\neg P_1(\overline{x}) \vee \neg F'$. Immediately $\neg P_1(\overline{x}) \vee G(\overline{x})$ is a $\mathsf{G}_{\approx}$ clause. Now consider $\neg P_1(\overline{x}) \vee \neg F'$. By i., F' is an existentially quantified sentence containing only flat and positive literals, hence $\neg F'$ is a universally quantified sentence containing only flat and negative literals. Since the existentially quantified variables in F' are universally quantified variables in $\neg F'$, $\neg F'$ may contain more variables than $\overline{x}$. By *3.–5.* of the **Trans**$^{\mathsf{GNF}}$ process, $\neg P(\overline{x}) \vee \neg F'$ is transformed into either a $\mathsf{Q}_{\approx}$ clause (if no conjunction occurs in $\neg F'$), or a set of $\mathsf{Q}_{\approx}$ clauses (if conjunctions occur in $\neg F'$).

iii.: Consider F_2. By *3.* of the **Trans**$^{\mathsf{GNF}}$ process, $\forall \overline{x}(G(\overline{x}) \wedge \neg F' \rightarrow P_2(\overline{x}))$ is transformed to $\forall \overline{x}(\neg G(\overline{x}) \vee F' \vee P_2(\overline{x}))$. W.l.o.g. suppose $\forall \overline{x}(\neg G(\overline{x}) \vee F' \vee P_2(\overline{x}))$ is transformed to C (if no conjunction occurs in F'), or transformed to $C_1, \ldots, C_n$ (if conjunctions occur in F'). For each C_i in $C_1, \ldots, C_n$, $\neg G(\overline{x})$ is the guard. The existential quantified variables in F' are skolemised into $f(\overline{x})$ where f is a Skolem function. Hence C_i is covering. By the fact that F_2 contains no function symbols, C_i is simple. By *3.* in **Definition** 22, C_i is a $\mathsf{G}_{\approx}$ clause. Note that if an equality literal $x \not\approx y$ is the only guard in a flat C_i, then by the equality resolution rule, $x \not\approx y \vee F' \vee P(x, y)$ is simplified into positive, flat and single-variable clauses $F' \vee P(x, x)$. By *2.* of **Definition** 22, C_i is a $\mathsf{G}_{\approx}$ clause. □

Theorem 7.1. *The* ***Trans***$^{\mathsf{GNF}}$ *process reduces the problem of* BCQ *answering for* GNF *to that of deciding satisfiability of the* $\mathsf{GQ}_{\approx}$ *clausal class.*

Proof. By **Lemma** 6.2 and the fact that the **Trans**$^{\mathsf{GNF}}$ process transforms a union of BCQs to a set of query clauses. □

Transforming CGNF to the $\mathsf{LGQ}_{\approx}$ clausal class

Next, we present the structural transformation that transforms clique guarded negation formulas, with a detailed example.

Recall the definition of CGNF from **Section** 2.1.

Definition 6. *The* clique guarded negation fragment (CGNF) *is a fragment of* $\mathsf{FOL}_{\approx}$ *without functional symbols, inductively defined as follows:*

1. $\top$ *and* $\bot$ *belong to* CGNF.
2. *If* A *is an atom, then* A *belongs to* CGNF.
3. *If* A *and* B *are atoms, then* $A \vee B$ *and* $A \wedge B$ *belong to* CGNF.
4. *If* F *belongs to* CGNF, *then* $\exists \overline{x} F$ *belongs to* CGNF.
5. *Let* F *be a clique guarded negation formula and* $\mathbb{G}(\overline{x}, \overline{y})$ *a conjunction of atoms. Let* $\overline{z}$ *denote the free variables of* F. *Then* $\exists \overline{x} \mathbb{G}(\overline{x}, \overline{y}) \wedge \neg F$ *belongs to* CGNF *if*
   (a) $\overline{z}$ *is a subset of* $\overline{y}$, *and*
   (b) *each variable in* $\overline{x}$ *occurs in only one atom of* $\mathbb{G}(\overline{x}, \overline{y})$, *and*
   (c) *each pair of distinct variables in* $\overline{y}$ *co-occurs in an atom of* $\exists \overline{x} \mathbb{G}(\overline{x}, \overline{y})$.

We use the notation **Trans**$^{\textsf{CGNF}}$ to denote the procedure of transforming clique guarded negation formulas and a union of BCQs with equality. The **Trans**$^{\textsf{CGNF}}$ process first negates the given union of BCQs with equality, obtaining a set of $Q_{\approx}$ *clauses*. In the second step, the **Trans**$^{\textsf{CGNF}}$ process transform clique guarded negation formulas to their clausal normal forms. We use the clique guarded negation formula

$$F = \left[\begin{array}{ll} \neg \exists x_1 x_2 x_3 (& \exists y_1 y_2 (A_1(x_1, x_2, y_1) \wedge A_1(x_2, x_3, y_2) \wedge A(x_1, x_3)) \wedge \\ & \neg \exists x_4 (B(x_1, x_2, x_4) \wedge B(x_2, x_3, x_4)) \qquad\qquad) \end{array} \right].$$

to show the computation of the **Trans**$^{\textsf{CGNF}}$ process, given as follows. Note that F is implicitly (clique) guarded by $\top$.

1. Add existential quantifications for free variables in F, and then apply the **Miniscoping** rules to the clique guards of F, obtaining

$$F_1 = \left[\begin{array}{ll} \neg \exists x_1 x_2 x_3 (& \exists y_1 A_1(x_1, x_2, y_1) \wedge \exists y_2 A_1(x_2, x_3, y_2) \wedge A(x_1, x_3) \wedge \\ & \neg \exists x_4 (B(x_1, x_2, x_4) \wedge B(x_2, x_3, x_4)) \qquad\qquad) \end{array} \right].$$

2. Apply the **Trans** rules to F_1, introducing fresh predicate symbols P (and respective literals $P(\ldots)$) to replace the clique guarded negation patterns $\exists \overline{x} \mathbb{G}(\overline{x}, \overline{y}) \wedge \neg F'$ in F_1, obtaining

$$F_2 = \left[p \wedge (\neg \exists x_1 x_2 x_3 P(x_1, x_2, x_3) \vee \neg p) \wedge F_2' \right]$$

where

$$F'_2 = \forall x_1x_2x_3((\exists y_1 A_1(x_1,x_2,y_1) \wedge \exists y_2 A_1(x_2,x_3,y_2) \wedge A(x_1,x_3) \wedge \\ \neg\exists x_4(B(x_1,x_2,x_4) \wedge B(x_2,x_3,x_4))) \rightarrow P(x_1,x_2,x_3)).$$

We say that

- p is the *replacing formula of* F_1, and
- $\neg\exists x_1x_2x_3P(x_1,x_2,x_3) \vee \neg p$ and F'_2 are the *definition formulas of* p and P, respectively.

3. Apply the **NNF** rules to F_2 to transform it to negation normal form, obtaining

$$F_3 = \left[p \wedge (\neg\exists x_1x_2x_3P(x_1,x_2,x_3) \vee \neg p) \wedge F'_3\right]$$

where

$$F'_3 = \forall x_1x_2x_3(\forall y_1\neg A_1(x_1,x_2,y_1) \vee \forall y_2\neg A_1(x_2,x_3,y_2) \vee \neg A(x_1,x_3) \vee \\ \exists x_4(B(x_1,x_2,x_4) \wedge B(x_2,x_3,x_4)) \vee P(x_1,x_2,x_3))$$

4. Apply the **Trans** rules to F_3, introducing fresh predicate symbols P' (and respective negative literals $\neg P'(\ldots)$) to replace universally quantified formulas in the clique guards of F_3, obtaining

$$F_4 = \left[\begin{array}{lll} & p\wedge & \\ (\neg\exists x_1x_2x_3 & P(x_1,x_2,x_3) \vee \neg p)\wedge & \\ \forall x_1x_2x_3(& \neg P'_1(x_1,x_2) \vee \neg P'_2(x_2,x_3) \vee \neg A(x_1,x_3) \vee \exists x_4(& \\ & B(x_1,x_2,x_4) \wedge B(x_2,x_3,x_4)) \vee P(x_1,x_2,x_3) &)\wedge \\ \forall x_1x_2(& P'_1(x_1,x_2) \vee \forall y_1\neg A_1(x_1,x_2,y_1) &)\wedge \\ \forall x_2x_3(& P'_2(x_2,x_3) \vee \forall y_2\neg A_1(x_2,x_3,y_2) &) \end{array}\right].$$

We say that

- $\forall x_1x_2x_3(\neg P'_1(x_1,x_2) \vee \neg P'_2(x_2,x_3) \vee \neg A(x_1,x_3) \vee \exists x_4(B(x_1,x_2,x_4) \wedge B(x_2,x_3,x_4)) \vee P(x_1,x_2,x_3))$ is the *replacing formula of* F'_3.
- $\forall x_1x_2(P'_1(x_1,x_2) \vee \forall y_1\neg A_1(x_1,x_2,y_1))$ is the *definition formula of* P'_1.
- $\forall x_2x_3(P'_2(x_2,x_3) \vee \forall y_2\neg A_1(x_2,x_3,y_2))$ is the *definition formula of* P'_2.

5. Transform formulas in F_4 to prenex normal form, and then apply Skolemisation using a Skolem function symbol f, obtaining

$$F_5 = \begin{bmatrix} & p \wedge F_5' & \wedge \\ \forall x_1 x_2 x_3 & \neg P(x_1, x_2, x_3) \vee \neg p & \wedge \\ \forall x_1 x_2 y_1(& P_1'(x_1, x_2) \vee \neg A_1(x_1, x_2, y_1) &)\wedge \\ \forall x_2 x_3 y_2(& P_2'(x_2, x_3) \vee \neg A_1(x_2, x_3, y_2) &) \end{bmatrix}.$$

where

$$F_5' = \forall x_1 x_2 x_3(\neg P_1'(x_1, x_2) \vee \neg P_2'(x_2, x_3) \vee \neg A(x_1, x_3) \vee$$
$$(B(x_1, x_2, f(x_1, x_2, x_3)) \wedge B(x_2, x_3, f(x_1, x_2, x_3))) \vee P(x_1, x_2, x_3))$$

6. Finally, apply the **CNF** rules to F_5 to transform it to conjunctive normal form and drop all universal quantifiers, obtaining a clausal set

$$p, \qquad \neg P(x_1, x_2, x_3) \vee \neg p,$$
$$P_1'(x_1, x_2) \vee \neg A_1(x_1, x_2, y_1), \qquad P_2'(x_2, x_3) \vee \neg A_1(x_2, x_3, y_2),$$
$$\neg P_1'(x_1, x_2) \vee \neg P_2'(x_2, x_3) \vee \neg A(x_1, x_3) \vee B(x_1, x_2, f(x_1, x_2, x_3)) \vee P(x_1, x_2, x_3),$$
$$\neg P_1'(x_1, x_2) \vee \neg P_2'(x_2, x_3) \vee \neg A(x_1, x_3) \vee B(x_2, x_3, f(x_1, x_2, x_3)) \vee P(x_1, x_2, x_3)$$

By the **Trans**$^{\textsf{CGNF}}$ process, a clique guarded negation formula is transformed into a set of *loosely guarded clauses with equality* and *query clauses with equality*. The loosely guarded clauses with equality are formally defined as follows.

Definition 24. *A* loosely guarded clause with equality ($\mathsf{LG}_{\approx}$ clause) *C is a simple and covering clause that may contain equality, satisfying the following conditions:*

1. *C is either ground, or*
2. *C is a positive and single-variable clause, or*
3. *C contains a negative flat subclause* $\neg G_1 \vee \ldots \vee \neg G_n$ *such that each variable pair in C co-occurs in a literal of* $\neg G_1 \vee \ldots \vee \neg G_n$.

In 3. **Definition 24**, the negative flat subclause $\neg G_1 \vee \ldots \vee \neg G_n$ is call the *loose guard* of the clause C.

We use the notation $\mathsf{LGQ}_{\approx}$ to denote the class of $\mathsf{LG}_{\approx}$ clauses and $\mathsf{Q}_{\approx}$ clauses. The **Trans**$^{\textsf{CGNF}}$ process transforms a clique guarded negation formula to a set of $\mathsf{LGQ}_{\approx}$ clauses, formally stated as:

Lemma 7.2. *Applying the* $\textbf{Trans}^{\textsf{CGNF}}$ *process to a clique guarded negation formula transforms it to a* $\textsf{LGQ}_{\approx}$ *clausal set.*

Proof. Suppose F is a clique guarded negation formula. Suppose that in *2.* of the $\textbf{Trans}^{\textsf{CGNF}}$ process, F' is the replacing formula for F, and F_1 and F_2 are the definition formulas for $\forall \overline{x}(P_1(\overline{x}) \rightarrow \exists \overline{y}\mathbb{G}(\overline{x}, \overline{y}) \wedge \neg F')$ and $\forall \overline{x}(\exists \overline{y}\mathbb{G}(\overline{x}, \overline{y}) \wedge \neg F' \rightarrow P_1(\overline{x}))$ with P_1 and P_2 fresh predicate symbols, respectively. By the fact that F' is an existentially quantified sentence containing only flat and positive literals, *3.–6.* in the $\textbf{Trans}^{\textsf{CGNF}}$ process transform F' into a ground and flat clause (if no conjunction occurs in F'), or a set of ground and flat clauses (if conjunctions occur in F'). In either case, F' is transformed into $\textsf{LGQ}_{\approx}$ clauses. Now we distinguish cases of F_1 and F_2.

F_1: By *3.–4.* of the $\textbf{Trans}^{\textsf{CGNF}}$ process, F_1 is transformed into $\neg P_1(\overline{x}) \vee \exists \overline{y}\mathbb{G}(\overline{x}, \overline{y})$ and $\neg P_1(\overline{x}) \vee \neg F'$. We first consider $\neg P_1(\overline{x}) \vee \neg F'$. By the fact that F' is an existentially quantified sentence containing only flat and positive literals, $\neg F'$ is a universally quantified sentence containing only flat and negative literals. Then $\neg P_1(\overline{x}) \vee \neg F'$ is transformed into a (set of) $\textsf{Q}_{\approx}$ clauses (if conjunctions occur in $\neg F'$). Next, we consider $\neg P_1(\overline{x}) \vee \exists \overline{y}\mathbb{G}(\overline{x}, \overline{y})$. Since $\exists \overline{y}\mathbb{G}(\overline{x}, \overline{y})$ is a conjunction of atoms, the **CNF** rules transform $\neg P_1(\overline{x}) \vee \exists \overline{y}\mathbb{G}(\overline{x}, \overline{y})$ into a set of clauses. Suppose C is one of these clauses. Then for each existential quantified variable y in C (if y exists), the prenex normal form and then applying the **Skolem** rule transform y to a compound term $f(\overline{x})$ where f is a Skolem symbol. Hence, C is covering. By the fact that $\neg P_1(\overline{x}) \vee \exists \overline{y}\mathbb{G}(\overline{x}, \overline{y})$ contains no function symbol, C_1 is simple. By the definition of structural transformation, $\neg P_1(\overline{x})$ is the guard for C. Hence C is an $\textsf{LGQ}_{\approx}$ clause.

F_2: By *3.* of the $\textbf{Trans}^{\textsf{CGNF}}$ process, F_2 is transformed into $\forall \overline{x}(\forall \overline{y}\neg\mathbb{G}(\overline{x}, \overline{y}) \vee F' \vee P_1(\overline{x}))$. Suppose *4.* of the $\textbf{Trans}^{\textsf{CGNF}}$ process introduce a predicate symbol P'. W.l.o.g. further suppose $\forall y \neg L(x, y)$ is a literal in $\forall \overline{y}\neg\mathbb{G}(\overline{x}, \overline{y})$ where $x \in \overline{x}$ and $y \in \overline{y}$, and F_2' is the replacing formula of F_2, and $P'(x) \vee \neg L(x, y)$ is the definition of $P'(x)$. Then immediately $P'(x) \vee \neg L(x, y)$ is an $\textsf{LGQ}_{\approx}$ clause. We use $\forall \overline{x}(\neg\mathbb{G}_1(\overline{x}) \vee F' \vee P_1(\overline{x}))$ to denote F_2', and hence F_2' can be presented as

$$\forall \overline{x}(\neg G_1(\ldots) \vee \ldots \vee \neg P'(\ldots) \ldots \vee \neg G_n(\ldots) \vee F' \vee P_1(\overline{x})),$$

where $\neg G_1(\ldots) \vee \ldots \vee \neg P'(\ldots) \ldots \vee \neg G_n(\ldots)$ represents $\neg\mathbb{G}_1(\overline{x})$. Note that F' is an existentially quantified sentence containing only flat and positive literals.

If there exists conjunctions in F', then the **CNF** rules transform F'_2 to a set of clauses. Suppose C is one of these clauses. By **Definition 6**, the set of negative literals $\neg\mathbb{G}_1(\overline{x})$ is the loose guard of C. For any existential quantified variable in F', it is skolemised into a compound term containing variables $\overline{x}$, hence C is covering. As $\forall \overline{x}\overline{y}(\neg\mathbb{G}_1(\overline{x}, \overline{y}) \vee F' \vee P_1(\overline{x}))$ contains no function symbols, C is simple. Hence C is an $\mathsf{LGQ}_{\approx}$ clause. □

One can also use the **Sep** rule to handle existential quantifications in the clique guard of clique guarded negation formulas. This fact follows from the discussion for the **Trans$^{\mathsf{CGF}}$** process, from **Section 6.1**.

Now we give the main result of this section.

Theorem 7.2. *The **Trans$^{\mathsf{CGNF}}$** process reduces the problem of BCQ answering for CGNF to that of deciding satisfiability of the $\mathsf{LGQ}_{\approx}$ clausal class.*

Proof. By **Lemma 7.2** and the fact that **Trans$^{\mathsf{CGNF}}$** process transforms a union of BCQs to $\mathsf{Q}_{\approx}$ clauses. □

The $\mathsf{LGQ}_{\approx}$ clausal class strictly subsumes the $\mathsf{GQ}_{\approx}$ clausal class, since by restricting the number of literals in a loose guard in $\mathsf{LGQ}_{\approx}$ clauses to one, one obtains a $\mathsf{GQ}_{\approx}$ clause. Hence in the next sections, we focus on deciding satisfiability of the $\mathsf{LGQ}_{\approx}$ clausal class.

7.2 The superposition-based top-variable system

In this section, we first give the basis of a saturation-based superposition inference system. Then based on this system, we given the superposition-based top-variable system, specifically devised for deciding satisfiability of the $\mathsf{LGQ}_{\approx}$ clausal class.

A saturation-based superposition inference system

We use the notation **T-Inf$_{\approx}$** to denote our superposition-based **P-Res** system for first-order clausal logic with equality.

The **Inf$_{\approx}$** system is the combination of the **Deduce**, the **E-Fact**, the **E-Res** and the **Para** rules from the **Satu$_{\approx}$** system (from **Section 3.4**) and the **P-Res**, the **Fact**, the **Delete** rules from the **Inf** system (from **Section 4.2**). In particular admissible orderings $\succ$ are extended to the multiset ordering $\succ^m$ as follows.

Non-equational literals $P(t_1, \ldots, t_n)$ are treated as $P(t_1, \ldots, t_n) \approx \mathbf{tt}$ where $\mathbf{tt}$ is a distinguished constant. By $\succ$ it is always the case that $\mathbf{tt}$ is the minimal constant. Positive equality literals $s \approx t$ are regard as $\{s, t\}$ and negative equality literals $s \not\approx t$ are regard as $\{s, t, \mathbf{tt}\}$, respectively. This extension is also given in the paramodulation calculus in **Section** 3.4.

Recall the **Satu**$_{\approx}$ system from **Section** 3.4. A derivation is computed using

The Deduce rule (for clauses with equality)

$$\frac{N}{N \cup \{C\}}$$

if C is a conclusion of either the **Fact**, or the **P-Res**, or the **E-Fact**, or the **E-Res** or the **Para** rule of clauses in N.

Conclusions of the equality factoring rule is computed using

The E-Fact rule

$$\frac{t_1 \approx u \vee t_2 \approx v \vee D}{(u \not\approx v \vee t_1 \approx v \vee D)\sigma}$$

if the following conditions are satisfied.

1. Nothing is selected in D and $(t_1 \approx u)\sigma$ is $\succ^m$-maximal with respect to $(t_2 \approx v \vee D)\sigma$.
2. $u\sigma \not\succeq t_1\sigma$.
3. $\sigma = \mathrm{mgu}(t_1 \doteq t_2)$.

Conclusions of the equality resolution rule is computed using

The E-Res rule

$$\frac{t_1 \not\approx t_2 \vee D}{D\sigma}$$

if the following conditions are satisfied.

1. Either $(t_1 \not\approx t_2)\sigma$ is selected or it is $\succ^m$-maximal with respect to $D\sigma$.
2. $\sigma = \mathrm{mgu}(t_1 \doteq t_2)$.

Conclusions of the ordered paramodulation rule is computed using

The Para rule

$$\frac{t_1 \approx u \vee D_1 \qquad L[t_2] \vee D_2}{(L[u] \vee D_1 \vee D_2)\sigma}$$

if the following conditions are satisfied.

1. Nothing is selected in $D_1\sigma$ and $(t_1 \approx u)\sigma$ is strictly $\succ^m$-maximal with respect to $D_1\sigma$.
2. If $L[t_2]$ is positive, $L[t_2]\sigma$ is strictly $\succ^m$-maximal with respect to $D_2\sigma$, or else $L[t_2]\sigma$ is either selected or $\succ^m$-maximal with respect to $D_2\sigma$.
3. t_2 is not a variable.
4. $u\sigma \not\succeq t_1\sigma$.
5. $\sigma = \text{mgu}(t_1 \doteq t_2)$.
6. Premises are variable disjoint.

Recall that in the **Para** rule, the premises $t_1 \approx u \vee D_1$ and $L[t_2] \vee D_2$ are called the *left premise* and the *right premise*, respectively.

Theorem 7.3. *The* ***Inf***$_\approx$ *system is sound and refutationally complete for general first-order clausal logic with equality.*

Proof. By **Theorems** 3.4 and 4.2. □

The top-variable superposition system

In this section, we give a new top-variable refinement and a new superposition-based top-variable inference system for the $\mathsf{LGQ}_\approx$ clauses. We use the notation $\textbf{T-Ref}_\approx^{\mathsf{LGQ}_\approx}$ to denote this new superposition-based top-variable refinement, and use the notation $\textbf{T-Inf}_\approx^{\mathsf{LGQ}_\approx}$ to denote the $\textbf{Inf}_\approx$ system endowed with the $\textbf{T-Ref}_\approx^{\mathsf{LGQ}_\approx}$ refinement.

Like the $\textbf{T-Ref}^{\mathsf{GQ}}$ and the $\textbf{T-Ref}^{\mathsf{LGQ}}$ refinements, the $\textbf{T-Ref}_\approx^{\mathsf{LGQ}_\approx}$ refinement uses any admissible ordering with a precedence in which function symbols are larger than constant, which are larger than predicate symbols. With this precedence, a lexicographic path ordering $\succ_{lpo}$ is used as an example. However, unlike the $\textbf{T-Ref}^{\mathsf{GQ}}$ and the $\textbf{T-Ref}^{\mathsf{LGQ}}$ refinements, the $\textbf{T-Ref}_\approx^{\mathsf{LGQ}_\approx}$ refinement extends $\succ_{lpo}$ with a multiset ordering to consider equality literals, which is given in the previous section. We use $\succ_{lpo}^m$ to denote this ordering refinement.

Algorithm 17 determines the eligible literal, or the **P-Res** eligible literals (with respect to a **Res** inference step), to an $\mathsf{LGQ}_\approx$ clause.

Algorithm 17: Determining the (**P-Res**) eligible literals for $\mathsf{LGQ}_{\approx}$ clauses

Input: A $\mathsf{LGQ}_{\approx}$ clausal set N and a clause C in N
Output: The (**P-Res**) eligible literals in C

1 **if** C *is a ground clause* **then**
2   **return** $\mathrm{Max}(C)$
3 **else if** C *has negatively occurring compound-term literals* **then**
4   **return** $\mathrm{SelectNC}(C)$
5 **else if** C *has positively occurring compound-term literals* **then**
6   **return** $\mathrm{Max}(C)$
7 **else if** C *contains negatively occurring equality literals* **then**
8   **return** $\mathrm{SelectNE}(C)$
9 **else if** C *is a flat and single-variable positive clause* **then**
10   **return** $\mathrm{Max}(C)$
11 **else** **return** $\mathrm{PResT}(N, C)$

Different from the **T-Ref**$^{\mathsf{LGQ}}$ refinement, the **T-Ref**$_{\approx}^{\mathsf{LGQ}_{\approx}}$ refinement (Lines 7–10) considers LGQ clauses with equality literals occurring. The following functions are new to find eligible literals in the $\mathsf{LGQ}_{\approx}$ clauses.

- $\mathrm{Max}(C)$ returns the (strictly) $\succ_{lpo}^{m}$-maximal literal with respect to the clause C.
- $\mathrm{SelectNE}(C)$ selects one of the inequality literals in the clause C.

In the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement, one can use *a priori checking* for the (strictly) maximal literals. This statement is formally supported by:

Lemma 7.3. *Under the restrictions of the* **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ *refinement, if an eligible literal L is (strictly) $\succeq_{lpo}^{m}$-maximal with respect to an $\mathsf{LGQ}_{\approx}$ clause C, then $L\sigma$ is (strictly) $\succeq_{lpo}^{m}$-maximal with respect to $C\sigma$, for any substitution σ.*

Proof. By the covering property of $\mathsf{LGQ}_{\approx}$ clauses and **Lemma 4.6**. □

Theorem 7.4. *The* **T-Inf**$_{\approx}^{\mathsf{LGQ}_{\approx}}$ *system is sound and refutationally complete for general first-order clausal logic with equality.*

Proof. By **Theorem 7.3** and the fact that the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement consists of admissible orderings with selection functions, and a specific form of the **P-Res** rule (the top-variable resolution rule). □

7.3 Deciding the $\mathsf{LGQ}_{\approx}$ clausal class

Deciding satisfiability of the $\mathsf{LG}_{\approx}$ clausal class

In this section, our aim is to prove that the **T-Inf**$_{\approx}^{\mathsf{LGQ}_{\approx}}$ system decides satisfiability of the $\mathsf{LG}_{\approx}$ clausal class. In particular we focus on the inference steps that are not included in the result from **Section 6.3**, i.e. the applications of the **Fact** and the **P-Res** rules to flat, single-variable and positive $\mathsf{LG}_{\approx}$ clauses, and the applications of the **E-Fact**, the **E-Res** or the **Para** rules to $\mathsf{LG}_{\approx}$ clauses.

We investigate the applications of the **Fact** rule to $\mathsf{LG}_{\approx}$ clauses, starting with the following supporting lemma.

Lemma 7.4. *Let $C = D \vee B$ be an $\mathsf{LG}_{\approx}$ clause with B a compound-term literal. Let σ be a substitution that substitutes all variables in C with either constants or variables. Then $D\sigma$ is an $\mathsf{LG}_{\approx}$ clause.*

Proof. When C is a single-variable positive clause, the statement trivially holds. The results for the rest of cases of C can be obtained by adapting 'guarded clauses' to '$\mathsf{LG}_{\approx}$ clauses' in **Lemma 4.12**. □

Now we consider the conclusions of applying the **Fact** rule to $\mathsf{LG}_{\approx}$ clauses.

Lemma 7.5. *Applying the **Fact** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement) to $\mathsf{LG}_{\approx}$ clauses derives $\mathsf{LG}_{\approx}$ clauses.*

Proof. When C is a positive single-variable clause, this lemma trivially holds. By **Lemma 7.4**, the results of rest of cases follow from **Lemma 6.7**. □

Next we consider the application of the **E-Fact** rule to $\mathsf{LG}_{\approx}$ clauses.

Lemma 7.6. *Applying the **E-Fact** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement) to $\mathsf{LG}_{\approx}$ clauses derives $\mathsf{LG}_{\approx}$ clauses.*

Proof. Recall the **E-Fact** rule (with a priori checking for maximality and the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement).

$$\frac{t_1 \approx u \vee t_2 \approx v \vee D}{(u \not\approx v \vee t_1 \approx v \vee D)\sigma}$$

if the following conditions are satisfied.

1. Nothing is selected in D and $t_1 \approx u$ is $\succ^m_{lpo}$-maximal with respect to $t_2 \approx v \vee D$.
2. $u \not\succeq^m_{lpo} t_1$.
3. $\sigma = \mathrm{mgu}(t_1 \doteq t_2)$.

In the application of the **E-Fact** rule, suppose an $\mathsf{LG}_{\approx}$ clause C is the premise $t_1 \approx u \vee t_2 \approx v \vee D$ and C' is the conclusion $(u \not\approx v \vee t_1 \approx v \vee D)\sigma$ where $\sigma = \mathrm{mgu}(t_1 \doteq t_2)$. By **Algorithm 17**, C satisfies either Line 1, or 5 or 9. We distinguish these cases.

Line 1: When C is ground and simple, it is immediate that C' is ground and simple. Thus C' is an $\mathsf{LG}_{\approx}$ clause.

Line 5: The premise C contains positively occurring compound-term literals, and no negatively occurring compound-term literals. Now suppose C is a single-variable, positive compound-term clause. By **Lemma 4.5** and 1.–2. of the **E-Fact** rule, t_1 is a compound term. By the covering property and 3. of the **E-Fact** rule, t_2 is a compound term. Then by 1.–2. of the **E-Fact** rule, u and v are variables. By the fact that C is a single-variable clause, σ is void and then C' is a single-variable $\mathsf{LG}_{\approx}$ clause with the loose guard $u \not\approx v$. Hence, C' is an $\mathsf{LG}_{\approx}$ clause. Next suppose C is an $\mathsf{LG}_{\approx}$ clause satisfying *3.* of **Definition 24**. By **Lemma 4.5** and 1.–2. of the **E-Fact** rule, t_1 is a compound term. By the fact that in a covering clause C, the presence of a ground compound in C means that C is ground, t_1 is a non-ground compound term. By the covering property, t_1 and t_2 are both non-ground compound terms, otherwise they are not unifiable. Suppose in C, $\mathbb{G}$ is the loose guard, L is a literal and t is a compound term. By **Definition 24**, $t_1 \approx u$ and $t_2 \approx v$ are simple and covering such that $\mathrm{var}(t_1 \approx u) = \mathrm{var}(t_2 \approx v) = \mathrm{var}(C)$. By **Lemma 4.10**, $(t_1 \approx u)\sigma$ and $(t_2 \approx v)\sigma$ are simple. Then $(u \not\approx v \vee t_1 \approx v)\sigma$ is simple. By *1.* in **Lemma 4.11** and the facts that L is simple and $\mathrm{var}(L) \subseteq \mathrm{var}(t_1 \approx u)$, $L\sigma$ is simple. This mean all literals in C' is simple, hence C' is simple. By the covering property, $\mathrm{var}(C) = \mathrm{var}(t_1) = \mathrm{var}(t)$. By *3.* in **Lemma 4.11**, $\mathrm{var}(C\sigma) = \mathrm{var}(t_1\sigma) = \mathrm{var}(t\sigma)$.

Hence C' is covering. By 3. of the **E-Fact** rule and the facts that t_1 and t_2 are flat non-ground compound terms such that $\mathrm{var}(t_1) = \mathrm{var}(t_2)$, the mgu σ substitutes variables in C with either variables or constants. Then by the fact that $\mathrm{var}(\mathbb{G}) = \mathrm{var}(C)$, $\mathrm{var}(\mathbb{G}\sigma) = \mathrm{var}(C\sigma)$ and $\mathbb{G}\sigma$ is flat, hence $C\sigma$ is loosely guarded by $\mathbb{G}\sigma$. Then C' is an $\mathsf{LG}_{\approx}$ clause.

Line 9: The premise C is a positive single-variable clause. When C is flat, the statement trivially holds. □

Next we look at the conclusions of applying the **Para** rule to $\mathsf{LG}_{\approx}$ clauses.

Lemma 7.7. *Applying the* **Para** *rule (endowed with the* **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ *refinement) to* $\mathsf{LG}_{\approx}$ *clauses derives* $\mathsf{LG}_{\approx}$ *clauses.*

Proof. Recall the **Para** rule with a priori checking for maximality and the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement.

$$\frac{t_1 \approx u \vee D_1 \qquad L[t_2] \vee D_2}{(L[u] \vee D_1 \vee D_2)\sigma}$$

if the following conditions are satisfied.

1. Nothing is selected in D_1 and $(t_1 \approx u)$ is strictly $\succ^m_{lpo}$-maximal with respect to D_1.
2. If $L[t_2]$ is positive, $L[t_2]$ is strictly $\succ^m_{lpo}$-maximal with respect to D_2, or else $L[t_2]$ is either selected or $\succ^m_{lpo}$-maximal with respect to D_2.
3. t_2 is not a variable.
4. $u \not\succeq^m_{lpo} t_1$.
5. $\sigma = \mathrm{mgu}(t_1 \doteq t_2)$.
6. Premises are variable disjoint.

Suppose $\mathsf{LG}_{\approx}$ clauses $C_1 = t_1 \approx u \vee D_1$ and $C_2 = L[t_2] \vee D_2$ are the premises in an application of the **Para** rule, producing a conclusion $C' = (L[u] \vee D_1 \vee D_2)\sigma$ with $\sigma = \mathrm{mgu}(t_1 \doteq t_2)$. By **Algorithm 14**, C_1 satisfies either Line 1, or Line 5 or Line 9, and C_2 satisfies either Line 1, or Line 3, or Line 5, Line 7 or Line 9. We distinguish cases of C_1

Line 1: Suppose C_1 is a ground and simple clause. By 1. and 3. of the **Para** rule, t_1 is either a constant or a ground and flat compound term. Suppose t_1

is a ground compound term. By 3. of the **Para** rule and the fact that t_1 and t_2 are unifiable, t_2 is a compound term. Since t_1 is ground, σ substitutes all variables of t_2 by constants. By the covering property, $\mathrm{var}(t_2) = \mathrm{var}(C_2)$. Hence σ substitutes all variables of C_2 by constants. Then C' is a simple and ground clause, namely is an $\mathsf{LG}_{\approx}$ clause. Next suppose t_1 is a constant. By 1. and 4. of the **Para** rule, C_1 is a flat and ground clause. By 3. of the **Para** rule, t_2 is a constant. Hence σ is void. By the facts that C_2 is an $\mathsf{LG}_{\approx}$ clause and C_1 is a flat and ground clause, C' is an $\mathsf{LG}_{\approx}$ clause.

Line 5: Suppose C_1 contains no negative compound-term literal, but contains positive compound-term literals. Then C satisfies either *2.* or *3.* of **Definition 24**. Suppose C is a compound-term single-variable positive clause. By 1. of the **Para** rule and **Lemma 4.5**, t_1 is a compound term. By 3. of the **Para** rule, t_2 is a compound term. By the covering property, $\mathrm{var}(t_2) = \mathrm{var}(C_2)$. Then either all variables of C_2 are substituted by the variable in C_1, or C_2 is ground, and the variable in C_1 is substituted by a constant. By the fact that C_2 is an $\mathsf{LG}_{\approx}$ clause and the mgu σ, the resolvent C' is an $\mathsf{LG}_{\approx}$ clause. Next suppose C satisfies *3.* of **Definition 24**. By **Lemma 4.5**, $t_1 \approx u$ is a compound-term literal. By 3. of the **Para** rule, t_2 is a compound term. By **Algorithm 17**, C_2 satisfies either Line 1, Line 5 or Line 7. Suppose C_2 is a ground compound-term clause (Line 1). Then t_2 is ground. By the covering property and the fact that t_1 and t_2 are unifiable, σ substitutes all variables of C_1 with constants. Then the resolvent C' is a simple and ground clauses, which is an $\mathsf{LG}_{\approx}$ clause. Suppose C_2 is a non-ground compound-term clause (Lines 5 and 7). Suppose in either C_1 or C_2, L is a simple literal and t is a compound term. Further suppose $\mathbb{G}$ is the loose guard of C_1. By the covering property and the fact that t_1 and t_2 are unifiable, σ substitutes all variables with either constants or variables. Then $L\sigma$ is simple and $\mathbb{G}\sigma$ is flat. By the covering property and the fact that $\mathrm{var}(t_1\sigma) = \mathrm{var}(t_2\sigma)$, $\mathrm{var}(C_1\sigma) = \mathrm{var}(C_2\sigma)$. By the fact that $\mathrm{var}(\mathbb{G}) = \mathrm{var}(C_1)$, $\mathrm{var}(\mathbb{G}\sigma) = \mathrm{var}(C_1\sigma) = \mathrm{var}(C_2\sigma)$. Then $\mathbb{G}\sigma$ the loose guard of C'. Since $\mathrm{var}(t_1) = \mathrm{var}(t) = \mathrm{var}(C_1)$ (or $\mathrm{var}(t_1) = \mathrm{var}(t) = \mathrm{var}(C_2)$), $\mathrm{var}(t\sigma) \subseteq \mathrm{var}(C_1\sigma)$ (or $\mathrm{var}(t\sigma) \subseteq \mathrm{var}(C_2\sigma)$). Then C' is covering. Hence, C' is an $\mathsf{LG}_{\approx}$ clause.

Line 9: Suppose C_1 is a flat and single-variable positive clause. Suppose t_1 is a constant. By 1. of the **Para** rule, C_1 is a flat ground clause. By the facts that C_2 is an $\mathsf{LG}_{\approx}$ clause and C_1 is a flat ground clause, the resolvent C' is an

LG$_\approx$ clause. Now suppose t_1 is a variable. By 3. of the **Para** rule, t_2 is either a constant or a compound term. Then σ substitutes the only variable in C_1 with either a constant or a compound term. Hence $C_1\sigma$ is either a flat ground clause, or a compound-term single-variable clause. Then by the facts that C_2 is an LG$_\approx$ clause and σ does not substitute variables in C_2, the resolvent C' is an LG$_\approx$ clause. □

Next we discuss the applications of the **E-Res** rule to LG$_\approx$ clauses.

Lemma 7.8. *Applying the **E-Res** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}_\approx}$ refinement) to an LG$_\approx$ clause derives an LG$_\approx$ clause.*

Proof. Recall the **E-Res** rule (with a priori checking for maximality and the **T-Ref**$^{\mathsf{LGQ}_\approx}$ refinement).

$$\frac{t_1 \not\approx t_2 \vee D}{D\sigma}$$

if the following conditions are satisfied.

1. Either $t_1 \not\approx t_2$ is selected or it is $\succ^m_{lpo}$-maximal with respect to D.
2. $\sigma = \text{mgu}(t_1 \doteq t_2)$.

Suppose C is the **E-Res** premise $t_1 \not\approx t_2 \vee D$ and C' is the **E-Res** conclusion $D\sigma$. By **Algorithm** 17, C satisfies either Line 1, or Line 3 or Line 7. We distinguish these cases as follows.

Line 1: When C is a simple ground LG$_\approx$ clause, the lemma trivially holds.

Line 3: The equality literal $t_1 \not\approx t_2$ contains compound terms. By the covering property, t_1 and t_2 are both compound terms, otherwise t_1 and t_2 are not unifiable. Then σ substitutes variables with variables and constants. By **Lemma** 7.4, $D\sigma$ is an LG$_\approx$ clause.

Line 7: The premise C is flat and $t_1 \not\approx t_2$ is selected by the SelectNE(C) function. Then the terms t_1 and t_2 are either variables or constants, and hence σ substitutes a variable in C with either a variable or a constant. By **Lemma** 7.4, $D\sigma$ is an LG$_\approx$ clause. □

Now we investigate the pairing property and the unification in applications of the **P-Res** rule to LG$_\approx$ clauses when multiple side premises occur.

Lemma 7.9. *In applications of the* **P-Res** *rule, endowed with the* **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ *refinement, to a flat clause* $C = \neg A_1 \vee \ldots \vee \neg A_m \vee \neg A_{m+1} \vee \ldots \vee \neg A_n \vee D$ *with* $\neg A_1 \vee \ldots \vee \neg A_m$ *as the top-variable literals and* $\mathsf{LG}_{\approx}$ *clauses* $C_1 = \neg B_1 \vee D_1, \ldots, C_m = \neg B_n \vee D_n$ *with* $m \leq n$*. Further suppose* B_i *and* B_j *are, respectively, compound-term literals and flat literals in* C_i *and* C_j *with* $1 \leq i \leq m$ *and* $1 \leq j \leq m$*. Then the following conditions hold.*

1. *In* $\neg A_i$*, top variables pair compound terms and non-top variables pair constants or variables.*
2. *All variables in* $\neg A_j$ *are top variables, pairing either constants or variables.*
3. *In* $\neg A_i$*, top variables* x *are unified with the compound terms pairing* x *(modulo variables that are substituted with either variables or constants), and non-top variables are unified with either constants or variables.*
4. *In* $\neg A_j$*, either all variables are unified with a common non-nested compound term and constants, or all variables are unified with variables and constants.*
5. *In* B_i*, all variables are unified with variables and constants.*
6. *In* B_j*, either all variables are unified with non-nested compound terms or all variables are unified with variables and constants.*
7. *Suppose a top variable* x *pairs a constant. Then in* C*, all negative literals are top-variable literals and all variables are unified with constants.*

Proof. Assume that σ and σ' are mgus such that $\sigma = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_m \doteq B_m)$ and $\sigma' = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$, respectively .

1.: This proof is similar to *1.* of **Lemma 4.13**. W.l.o.g. suppose $\neg A_i$ and B_i are in the forms of $\neg A_i(\ldots, x, \ldots, y, \ldots)$ and $B_i(\ldots, t_1, \ldots, t_2, \ldots)$, respectively. Further suppose in $\neg A_i(\ldots, x, \ldots, y, \ldots)$, x is a top-variable, y is a non-top variable, and x and y pair t_1 and t_2, respectively. We prove *1.* by contradiction.

First assume that t_1 is not a compound term. This implies that t_1 is either a variable or a constant. By the fact that B_i is a compound-term literal, w.l.o.g. we assume that in B_i, t occurs as a compound term, pairing a variable z in $\neg A_i$. By the covering property, $\mathrm{var}(t_1) \subset \mathrm{var}(t)$. Then by the fact that $\mathrm{dep}(t_1) < \mathrm{dep}(t)$, $\mathrm{dep}(t_1\sigma') < \mathrm{dep}(t\sigma')$, hence $\mathrm{dep}(x\sigma') < \mathrm{dep}(z\sigma')$. By the definition of variable orderings, the case $\mathrm{dep}(x\sigma') < \mathrm{dep}(z\sigma')$ contradicts the fact that x is a top variable. Hence t_1 is a compound term, pairing x. Next assume that t_2 is neither a variable nor a constant. Then t_2 is a compound term. By the covering property and the fact that t_1 is a compound term, $\mathrm{dep}(t_1\sigma') =$

dep($t_2\sigma'$), therefore dep($x\sigma'$) = dep($y\sigma'$). This contradicts the fact that y is non-top variable.

2.: By **Algorithm 17** and the fact that B_j is a flat literal, C_j is either a flat ground clause or a flat single-variable clause. Then *2.* follows from the fact that B_j is an eligible literal.

3.: By the pairing property proved in *1.* and *2.*.

4.: By **Algorithm 17**, we distinguish two cases of side premises.

4.-1: Assume that all side premises $C_1, \ldots, C_m$ are flat clauses. It is immediate that in $\neg A_j$, all variables are unified with variables and constants.

4.-2: Assume that both compound-terms clauses and flat clauses occur in the side premises $C_1, \ldots, C_m$. Suppose x and y are top variables in the compound-term literal B_i and the flat literal B_j, respectively. By *3.*, dep($x\sigma$) = 1. By the fact that x and y are both top variables, dep($y\sigma$) = 1. Hence y is unified with a non-nested compound-term. By **Algorithm 17**, B_j is a flat single-variable clause such that only a single variable and constants occur as its arguments. Suppose y is unified with the compound-term t. Then the only variable in B_j is unified with t. Hence, all variables in $\neg A_i$ are unified with a common non-nested compound term and constants.

5. and *6.*: By *3.* and *4.*, respectively.

7.: By the proof in *5.* of **Lemma 4.13**. □

Now we apply the top-variable resolution rule to $\mathsf{LG}_{\approx}$ clauses.

Lemma 7.10. *Applying the* **P-Res** *rule (endowed with the* **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ *refinement) to* $\mathsf{LG}_{\approx}$ *clauses derives* $\mathsf{LG}_{\approx}$ *clauses.*

Proof. By **Algorithm 17**, in applications of the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement) to $\mathsf{LG}_{\approx}$ clauses, the positive premise satisfies either Line 1 (it is ground), or Line 5 (it has positively occurring compound-term literals, but does not have negatively occurring compound-term literals), or Line 9 (it is a flat and single-variable positive clause). In this proof, we focus on the case when flat and single-variable positive clauses occur as side premises. Note that when a side premise is a compound-term single-variable positive clauses, these side premises are implicitly guarded by the inequality literal $x \not\approx x$. For the rest of cases when side premises satisfy Lines 1 and 5 in **Algorithm 17**, by *3.* and *5.* in **Lemma 7.9**, it follows from **Lemma 6.8** that applying the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement) to $\mathsf{LG}_{\approx}$ clauses derives $\mathsf{LG}_{\approx}$ clauses.

By **Algorithm 17**, in applications of the **P-Res** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement) to $\mathsf{LG}_{\approx}$ clauses, the negative premise satisfies either Line 1, or Line 3 or Line 11. Note that the **P-Res** rule is reduced to a *binary resolution rule* when the negative premise satisfies either Line 1 or Line 3. Suppose in applications of the **P-Res** rule, the negative premise $C = \neg A_1 \vee D$ is either a simple and ground clause (Line 1), or has negatively occurring compound-term literals (Line 3), the positive premise $C_1 = B_1 \vee D_1$ is a flat and single-variable positive clause and the resolvent is $C' = (D \vee D_1)\sigma$ with an mgu σ such that $\sigma = \mathrm{mgu}(A_1 \doteq B_1)$. As C_1 is flat, single-variable and positive, it is trivial that in either case the resolvent C' is an $\mathsf{LG}_{\approx}$ clause.

Next we consider the case when the negative premise satisfies Line 11 in **Algorithm 17**. Suppose in an application of the **P-Res** rule to $\mathsf{LG}_{\approx}$ clauses, the positive premises are $\mathsf{LG}_{\approx}$ clauses $C_1 = B_1 \vee D_1, \ldots, C_n = B_n \vee D_n$, the negative premise is a non-ground flat $\mathsf{LG}_{\approx}$ clause $C = \neg A_1 \vee \ldots \vee \neg A_m \vee \neg A_{m+1} \vee \ldots \vee \neg A_n \vee D$ (with D a positive subclause) and the resolvent is $C' = (D_1 \vee \ldots \vee D_m \vee \neg A_{m+1} \vee \ldots \vee A_n \vee D)\sigma$ with σ an mgu such that $\sigma = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_m \doteq B_m)$ where $m \leq n$. We distinguish two cases when flat and single-variable positive clauses occur in $C_1, \ldots, C_n$.

1.: All of clauses in $C_1, \ldots, C_n$ are flat and single-variable positive clauses. By the $\mathrm{CompT}(C_1, \ldots, C_n, C)$ function and the fact that $C_1, \ldots, C_n$ are flat, in C all variables in $\neg A_1 \vee \ldots \vee \neg A_n$ are top variables and $\neg A_1, \ldots, \neg A_n$ are the top-variable literals with $m = n$. By *4.* of **Lemma 7.9**, all variables in $\neg A_1 \vee \ldots \vee \neg A_n$ are unified with constants and variables. Now we consider the unification of variables in C. Suppose x_i and x_j are two variables occurring in $\neg A_1 \vee \ldots \vee \neg A_n$. Since C has the variable co-occurrence property, w.l.o.g. we assume that x_i and x_j co-occur in $\neg A_t$ with $1 \leq t \leq n$. As B_t is a flat and single-variable literal, $x_i\sigma$ is either a variable or a constant, and $x_i\sigma$ is identical to $x_j\sigma$. Hence all variables in $\neg A_1 \vee \ldots \vee \neg A_n$ are unified with a common variable and constants. By *3.* of **Definition 24** and the fact that C is an $\mathsf{LG}_{\approx}$ clause, $\mathrm{var}(\neg A_1 \vee \ldots \vee \neg A_n) = \mathrm{var}(C)$. Then all variables in C are unified with a common variable and constants. By the fact that $C_1, \ldots, C_n$ are single-variable clauses, all variables in $C_1, \ldots, C_n$ are unified with a common variable and constants. Since $C_1, \ldots, C_n$ and C are flat, C' is a flat clause containing no more than one variable. By the fact that all negative literals in C are resolved as top-variable literals and $C_1, \ldots, C_n$ are positive clauses, C' is positive. Then C' is an LG$_\approx$ clause.

2.: Both flat, single-variable positive clauses and compound-term clauses occur in $C_1, \ldots, C_n$. W.l.o.g. suppose C_i and C_j are, respectively, a compound-term clause and a flat, single-variable and positive clause with $1 \leq i \leq m$ and $1 \leq j \leq m$. By **Algorithm 17**, B_i and B_j are, respectively, a compound-term literal and a flat, single-variable and positive literal. We prove that the resolvent C' is an LG$_\approx$ clause by proving that C is simple, covering and has a loose guard. By the covering property and the fact that C_j contains only one variable, $\text{var}(B_i) = \text{var}(C_i)$ and $\text{var}(B_j) = \text{var}(C_j)$, respectively. By *3.–6.* of **Lemma 7.9**, C' is simple. Next we prove that C' is covering and contains a loose guard. Suppose $x_1, \ldots, x_{m'}$ are the set of top variables in C. Further suppose that x_i and x_j are top-variables in $x_1, \ldots, x_{m'}$, occurring in $\neg A_i$ and $\neg A_j$, respectively. By *3.* of **Lemma 7.9**, x_i is substituted by either a compound term or a constant that x_i pairs. First suppose x_i pairs a constant. By *7.* of **Lemma 7.9** and the fact that an eligible literals of the side premise in $C_1, \ldots, C_n$ shares the same variable set as that side premise, the resolvent C' is a flat ground clause, therefore C' is an LG$_\approx$ clause. Next suppose x_i pairs a compound term. By the variable co-occurrence property of C, further suppose x_i and x_j co-occur in a literal $\neg A_t$ of $\neg A_1 \vee \ldots \vee \neg A_m$. Suppose $C_t = B_t \vee D_i$ is that side premise such that A_t pairs B_t. By the covering property and *3.* of **Lemma 7.9**, $\text{var}(x_i\sigma) = \text{var}(x_j\sigma) = \text{var}(A_t\sigma) = \text{var}(B_t\sigma)$. By the variable co-occurrence property of C, $\text{var}(x_1\sigma) = \ldots = \text{var}(x_{m'}\sigma) = \text{var}((\neg A_1 \vee \ldots \vee \neg A_m)\sigma)$. By *3.* of **Definition 24**, x_i co-occurs with all other variable in C in $\neg A_1 \vee \ldots \vee \neg A_m$. Hence $\text{var}(C) = \text{var}(\neg A_1 \vee \ldots \vee \neg A_m)$. Then $\text{var}(C\sigma) = \text{var}(x_1\sigma) = \ldots = \text{var}(x_{m'}\sigma)$. By the covering property, $\text{var}(C_t) = \text{var}(B_t)$. Then $\text{var}(C\sigma) = \text{var}(C_t\sigma)$. Then we have $\text{var}(C\sigma) = \text{var}(C_i\sigma)$ for all i such that $1 \leq i \leq m$. Since C is a flat clause, compound terms in the resolvent C' are inherited from compound-term clauses in $D_1, \ldots, D_m$. Suppose $\mathbb{G}$ is a loose guard and t a compound term in a C_i of $C_1, \ldots, C_m$. By **Definition 24**, $\text{var}(t) = \text{var}(\mathbb{G}) = \text{var}(C)$. Then $\text{var}(t\sigma) = \text{var}(\mathbb{G}\sigma) = \text{var}(C\sigma) = \text{var}(C_i\sigma)$ for all i such that $1 \leq i \leq m$. By *3.–6.* of **Lemma 7.9**, $\mathbb{G}\sigma$ is flat and $t\sigma$ is a non-nested compound term. Hence, C' is simple, covering and has a loose guard $\mathbb{G}\sigma$, hence, C' is an LG≈ clause. □

In applications of the **T-Inf**$_\approx^{\mathsf{LGQ}_\approx}$ system to LG$_\approx$ clauses, the width of the derived LG$_\approx$ clauses are bounded, formally stated as:

Lemma 7.11. *In applications of the* $\textbf{\textit{T-Inf}}_{\approx}^{\mathsf{LGQ}_{\approx}}$ *system to* $\mathsf{LG}_{\approx}$ *clauses, the derived* $\mathsf{LG}_{\approx}$ *clause is no wider than at least one of its premises.*

Proof. The statement trivially holds when a conclusion is either a single-variable clause or a ground clause.

By **Lemmas** 7.5, 7.6 and 7.8, applying, respectively, the **Fact**, **E-Fact** and **E-Res** rules to $\mathsf{LG}_{\approx}$ clauses derives $\mathsf{LG}_{\approx}$ clauses C'. By the fact that the loose guard in C' is inherited from its premise, C' contains no more types of variables than its premise. By **Lemmas** 7.7 and 7.10, applying the **Para** and **P-Res** rules to $\mathsf{LG}_{\approx}$ clauses derives $\mathsf{LG}_{\approx}$ clauses C'. The loose guard in C' is inherited from the right premise in the **Para** inference and the side premise in the **P-Res** inference. Hence C' is no more wider than at least one of its premises. □

Theorem 7.5. *The* $\textbf{\textit{T-Inf}}_{\approx}^{\mathsf{LGQ}_{\approx}}$ *system decides satisfiability of the* $\mathsf{LG}_{\approx}$ *clausal class.*

Proof. By **Lemmas** 7.5–7.8 and 7.10, applying the rules in the $\textbf{T-Inf}_{\approx}^{\mathsf{LGQ}_{\approx}}$ system to $\mathsf{LG}_{\approx}$ clauses derives $\mathsf{LG}_{\approx}$ clauses. By the fact that $\mathsf{LG}_{\approx}$ clauses are simple, the depth of derived $\mathsf{LG}_{\approx}$ clauses is bounded by a constant. By **Lemma** 7.11, the width of derived $\mathsf{LG}_{\approx}$ clauses is also bounded by a constant. □

Handling $\mathsf{Q}_{\approx}$ clauses (in the presence of the $\mathsf{LG}_{\approx}$ clauses)

In this section, we compute inferences when a $\mathsf{Q}_{\approx}$ clause is the premise. Our aim is to eliminate inequality literals in $\mathsf{Q}_{\approx}$ clauses, reducing $\mathsf{Q}_{\approx}$ clauses to *query clauses*, which can be handled by the techniques in **Section** 4.5.

Since $\mathsf{Q}_{\approx}$ clauses are negative clauses, the **Fact** and the **E-Fact** rules are not applicable to them. By the fact that $\mathsf{Q}_{\approx}$ clauses are negative and flat, in the **Para** rule a $\mathsf{Q}_{\approx}$ clause cannot be a left premise and a right premise, respectively. Hence we focus on the applications of the **E-Res** and **P-Res** rules to $\mathsf{Q}_{\approx}$ clauses.

We start with considering applying the **E-Res** rule to $\mathsf{Q}_{\approx}$ clauses.

Lemma 7.12. *Applying the* **E-Res** *rule (endowed with the* $\textbf{\textit{T-Ref}}^{\mathsf{LGQ}_{\approx}}$ *refinement) to* $\mathsf{Q}_{\approx}$ *clauses derives* $\mathsf{Q}_{\approx}$ *clauses.*

Proof. Recall the **E-Res** rule (with a priori checking for maximality and the $\textbf{T-Ref}^{\mathsf{LGQ}_{\approx}}$ refinement).

$$\frac{t_1 \not\approx t_2 \vee D}{D\sigma}$$

if the following conditions are satisfied.

1. Either $t_1 \not\approx t_2$ is selected or it is $>^m_{lpo}$-maximal with respect to D.
2. $\sigma = \mathrm{mgu}(t_1 \doteq t_2)$.

Suppose the **E-Res** premise is a $\mathsf{Q}_{\approx}$ clause C such that $C = t_1 \not\approx t_2 \vee D$, and the conclusion is C' such that $C' = D\sigma$. By **Algorithm 17**, a $\mathsf{Q}_{\approx}$ clause satisfies either Line 1 or Line 7. The case is trivial when the premise C is a flat and negative ground clause. Line 7 in **Algorithm 17** requires the premise C to be a non-ground, flat and negative clause. Then the mgu σ substitutes a variable in C with either a constant or a variable. In either case, $D\sigma$ is a $\mathsf{Q}_{\approx}$ clause. □

Observed that by **Algorithm 17**, in the $\textbf{T-Inf}_{\approx}^{\mathsf{LGQ}_{\approx}}$ system only the **E-Res** rule is applicable to non-ground $\mathsf{Q}_{\approx}$ clauses with inequality literals occurring. By this observation, we can put our focus on equality-free $\mathsf{Q}_{\approx}$ clauses, i.e., *query clauses*.

Recall that in **Section 4.5** the **Q-Sep** procedure separates a *query clause* into *Horn guarded clauses* (*HG clauses*) and an *indecomposable chained-only query clause* (*indecomposable CO clause*). By **Algorithm 14**, the **P-Res** rule is applied to an indecomposable CO clause (as a main premise) and $\mathsf{LG}_{\approx}$ clauses (as side premises), deriving the top-variable resolvents R. We abusively reuse the notion **T-Trans** to denote the *formula renaming* technique that transforms R to query clauses and $\mathsf{LG}_{\approx}$ clauses. The only difference of this **T-Trans** rule and the **T-Trans** rule in **Section 4.5** is that the side premises are $\mathsf{LG}_{\approx}$ clauses, instead of guarded clauses.

We use notation $\textbf{Q-CO}^{\mathsf{LGQ}_{\approx}}$ to denote our procedure to handle indecomposable CO clauses in the presence of $\mathsf{LG}_{\approx}$ clauses, given as follows.

1. Apply the top-variable resolution rule to an indecomposable CO clause and $\mathsf{LG}_{\approx}$ clauses, deriving the top-variable resolvent R.

2. Apply the **T-Trans** rule to R, deriving a query clause Q and $\mathsf{LG}_{\approx}$ clauses.

3. Apply the **Q-Sep** procedure to Q, producing HG clauses and an indecomposable CO clause.

Lemma 7.13. *The conclusions of applying the* **Q-CO$^{\mathsf{LGQ}_{\approx}}$** *procedure to an indecomposable* CO *clause Q and a set N of* $\mathsf{LG}_{\approx}$ *clauses satisfy the following conditions.*

1. *The conclusions are an indecomposable* CO *clause Q' and a set N' of* $\mathsf{LG}_{\approx}$ *clauses.*
2. *The clausal sets $Q' \cup N'$ and $Q \cup N$ are equisatisfiable.*
3. *For each clause C' in N', there exists a clause C in N such that C' is no wider than C, and Q' is less wide than Q.*

Proof. By **Lemma 7.9** and the fact that flat and single-variable positive clauses occurring as the side premises of the **P-Res** rule does not hurt the result established in **Lemma 6.10**. □

7.4 Decision procedures for answering and rewriting BCQs for GNF and/or CGNF

BCQ answering for GNF and CGNF

In this section, we give the formal decision procedure for answering BCQs for GNF and/or CGNF.

Algorithm 18 gives the pre-process steps for the given (clique) guarded negation formulas and union of BCQs.

Algorithm 18: The PreProcessCGNF function

Input: A union q of BCQs, sets Σ_1 and Σ_2 of formulas in GNF and CGNF, respectively

Output: A set of $\mathsf{LGQ}_{\approx}$ clauses

```
Function PreProcessCGNF(Σ1, Σ2, q):
    usable ← ∅
    G≈, Q1≈ ← TransGNF(Σ1, q)
    LG≈, Q2≈ ← TransCGNF(Σ2, q)
    usable ← usable ∪ G≈ ∪ LG≈ ∪ Q1≈ ∪ Q2≈
    usable ← Red(usable, usable)
    return usable
```

Unlike the PreProcessCGF function from **Section 6.4**, **Algorithm 18** uses the following new functions.

1. TransGNF(Σ, q) applies the **Trans**$^{\mathsf{GNF}}$ process to a set Σ of guarded negation formulas and a union q of BCQs, outputting a set of $\mathsf{GQ}_{\approx}$ clauses.

2. TransCGNF(Σ, q) uses the **Trans**$^{\mathsf{CGNF}}$ process to a clique guarded negation formula set Σ and a union q of BCQs, outputting an $\mathsf{LGQ}_{\approx}$ clausal set.

3. PreProcessCGNF(Σ_1, Σ_2, q) takes a union q of BCQs, a set Σ_1 of formulas in GNF and a set Σ_2 of formulas in CGNF as input, returning an $\mathsf{LGQ}_{\approx}$ clausal set.

Based on the *give-clause algorithm*, **Algorithm 19** on the next page gives a sample decision procedure for answering a union of BCQs for GNF and/or CGNF. We use the notation **Q-Ans**$^{\mathsf{CGNF}}$ to denote the decision procedure in **Algorithm 19**. Compared to the **Q-Ans**$^{\mathsf{CGF}}$ procedure, the **Q-Ans**$^{\mathsf{CGNF}}$ procedure has the following new functions, specially for reasoning with the equality literals.

1. E-Fact(C) applies the **E-Fact** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement) to the clause C, outputting the conclusion of C.

2. E-Res(C) applies the **E-Res** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement) to the clause C, outputting the conclusion of C.

3. Para(C_1, C_2) applies the **Para** rule (endowed with the **T-Ref**$^{\mathsf{LGQ}_{\approx}}$ refinement) to the clauses C_1 and C_2, outputting the conclusion of C_1 and C_2.

Another major difference of the **Q-Ans**$^{\mathsf{CGF}}$ and the **Q-Ans**$^{\mathsf{CGNF}}$ procedures is that in the **Q-Ans**$^{\mathsf{CGNF}}$ procedure, the Sep function (the **Q-Sep** procedure) is applied in the saturation process in Lines 11–13 of **Algorithm 19**.

Lemma 7.14. *In the **Q-Ans**$^{\mathsf{CGNF}}$ procedure, only finitely many predicate symbols are introduced.*

Proof. It follows from the fact that allowing equality literals and single-variable positive clause in the LG clausal class does not hurt the results established in **Lemmas 4.27** and **6.12**. □

Algorithm 19: The BCQ answering procedure for GNF and CGNF

Input: A union q of BCQs and sets Σ_1 and Σ_2 of formulas in GNF and CGNF, respectively
Output: 'Yes' or 'No'

```
workedOff ← ∅
usable ← PreProcessCGNF(Σ1, Σ2, q)
while usable = ∅ and ⊥ ∉ usable do
    given ← Pick(usable)
    workedOff ← workedOff ∪ given
    if given is a query clause then
        CO, HG ← Sep(given)
        new ← CO ∪ HG
    if given is an indecomposable CO clause then
        tResolvent ← P-Res(workedOff, given)
        LG≈, Q ← T-Trans(tResolvent)
        CO, HG ← Sep(Q)
        new ← LG≈ ∪ CO ∪ HG
    else
        new ← P-Res(workedOff, given) ∪ Fact(given) ∪
          E-Fact(given) ∪ E-Res(given) ∪ Para(workedOff, given)
    new ← Red(new, new)
    new ← Red(Red(new, workedOff), usable)
    workedOff ← Red(workedOff, new)
    usable ← Red(usable, new) ∪ new
Print(usable)
```

Finally we give the first main result of this chapter, providing a positive answer to **Problem** 8.

Theorem 7.6. *The* **Q-Ans**$^{\mathsf{CGNF}}$ *procedure is a decision procedure for answering BCQs for UNF, GNF and/or CGNF.*

Proof. By **Theorems** 7.1–7.2 and the fact that UNF is a subfragment of GNF such that guards are restricted to the inequality literal $x \not\approx y$, the **Q-Ans**$^{\mathsf{CGNF}}$ procedure reduces the problem of answering BCQs for UNF, GNF and/or CGNF to that of deciding satisfiability of the $\mathsf{LGQ}_{\approx}$ clausal class. By **Lemma** 4.19 and **Theorem** 7.4, the $\textbf{T-Inf}_{\approx}^{\mathsf{LGQ}_{\approx}}$ system is a sound and refutationally complete system for general first-order clausal logic. As the $\textbf{Q-Ans}^{\mathsf{CGNF}}$ procedure is based on the $\textbf{T-Inf}_{\approx}^{\mathsf{LGQ}_{\approx}}$ system and our customised separation rules, the $\textbf{Q-Ans}^{\mathsf{CGNF}}$ procedure is a sound and refutational complete procedure if only finitely many predicate symbols are introduced.

By **Lemma** 4.23, **Lemma** 7.13 and **Theorem** 7.5, applying the $\textbf{Q-Ans}^{\mathsf{CGNF}}$ procedure to $\mathsf{LGQ}_{\approx}$ clauses guarantees producing $\mathsf{LGQ}_{\approx}$ clauses of bounded depth and bounded width. By **Lemma** 7.14, only finitely many new predicate symbols (with respect to the given signature) are introduced, hence the $\textbf{Q-Ans}^{\mathsf{CGNF}}$ procedure guarantees termination. Since the $\textbf{Q-Ans}^{\mathsf{CGNF}}$ procedure is sound, refutationally complete for first-order clausal logic and guarantees termination for the $\mathsf{LGQ}_{\approx}$ clausal class, it is a decision procedure for answering BCQs for UNF, GNF and/or CGNF. □

The $\textbf{Q-Ans}^{\mathsf{CGNF}}$ procedure can be altered by the following implementations:

1. In Lines 6–8 of **Algorithm** 19, the **Q-Sep** procedure can be extended to apply to the $\mathsf{Q}_{\approx}$ clausal class, instead of *query clauses*. Regarding equality literals as general binary literals, the result established in **Lemma** 4.23 can be easily generalised to $\mathsf{Q}_{\approx}$ clauses. However this alteration complicates the following $\textbf{Q-CO}^{\mathsf{LGQ}_{\approx}}$ procedure, since applying the **Q-Sep** procedure to $\mathsf{Q}_{\approx}$ clauses derives *indecomposable CO clause with equality*, which are not suitable premises for the top-variable resolution rule. For example, due to the occurrence of the equality literal $x \not\approx z$, the CO clause with equality $Q = \neg A(x,y) \vee \neg A(y,z) \vee x \not\approx z$ cannot be the main premise in the top-variable resolution rule. Hence this alteration may not be a wise choice. By the $\textbf{T-Inf}_{\approx}^{\mathsf{LGQ}_{\approx}}$ system, only the **E-Res** rule is applicable to clauses like Q, deriving new query clauses, which are then handled by the **Q-Sep** procedure. In this example, applying **E-Res** rule to Q derives an IO clause $\neg A(x,y) \vee \neg A(y,x)$, which is also an HG clause.

2. The applications of the **E-Res** rule to $\mathsf{Q}_{\approx}$ clauses in the saturation process (Lines 3–15 of **Algorithm** 19) can be moved to the PreProcessCGNF function. Note that i) by **Algorithm** 17, only the **E-Res** rule is applicable to non-ground $\mathsf{Q}_{\approx}$ clause, and ii) by **Lemma** 7.12, applying the **E-Res** rule to non-ground $\mathsf{Q}_{\approx}$ clauses only derives $\mathsf{Q}_{\approx}$ clauses. By i) and ii), one

can independently saturate the *non-ground* $\mathsf{Q}_{\approx}$ *clauses with equality literal occurring* by the **E-Res** rule. Suppose N is a $\mathsf{Q}_{\approx}$ clausal set. By the **E-Res** rule, N can be saturated to a set N_1 of *non-ground* $\mathsf{Q}_{\approx}$ *clauses with equality literal occurring*, a set N_2 of *ground* $\mathsf{Q}_{\approx}$ *clauses* and a set N_3 of *non-ground query clauses*. Though the clauses in N_2 and N_3 need to be considered in the saturation process (Lines 3–15 of **Algorithm 19**), the clauses in N_1 can be immediately added to the final saturated clausal set (the *workedOff* clausal set in Line 18 of **Algorithm 19**). This is due to the fact that in the **Q-Ans**$^{\mathsf{CGNF}}$ procedure, only the **E-Res** rule is applicable to clauses in N_1.

Rewriting BCQs for GNF and CGNF

In contrast to the saturation-based BCQ rewriting procedures for the *guarded quantification fragments* that a clausal set is back-translated to a *first-order formula*, in this section we tackle a more challenging task, that is the back-translation from a saturated clausal set of negated BCQs and (C)GNF to a (*clique*) *guarded negation formula*.

Unlike the classes of GQ^- and the LGQ^- clauses, the $\mathsf{LG}_{\approx}$ clausal class is further refined by the notion *protect*.

Definition 25. *A clause C is* protected *if all compound terms $t = (s_1, \ldots, s_m)$ in C satisfy the following conditions.*

1. *There exists a negative flat subclause $\neg G_1 \vee \ldots \vee \neg G_n$ in C such that each pair of arguments in t co-occurs in a literal of $\neg G_1 \vee \ldots \vee \neg G_n$, and*
2. *for each term s_i in $s_1, \ldots, s_m$, $\mathrm{Occ}(s_i, t) \leq \mathrm{Occ}(s_i, \neg G_1 \vee \ldots \vee \neg G_n)$.*

By adopting the notions of *strongly compatible* (from **Definition 12**) and *protect* to $\mathsf{LG}_{\approx}$ clauses, we formally define the *aligned* (*loosely*) *guarded clauses with equality*.

Definition 26. *An* aligned loosely guarded clause with equality ($\mathsf{LG}_{\approx}^-$ clause) *is an $\mathsf{LG}_{\approx}$ clause that is protected and strongly compatible.*

A aligned guarded clause with equality *is an $\mathsf{LG}_{\approx}^-$ clause with one negative flat literal as its loose guard.*

The *protect* property ensures that given an $\mathsf{LG}_{\approx}^-$ clause C, every argument in the compound terms of C is mapped to an argument in its loose guard.

Note that if an $\mathsf{LG}^{-}_{\approx}$ clause C is a positive single-variable clause, C is also *protected* as C is implicitly guarded by the inequality literal $x \not\approx x$.

Lemma 7.15. *Applying the* ***Trans***$^{\mathsf{GNF}}$ *process to a guarded negation formula transforms it into a set of aligned guarded clauses with equality, and ii) applying the* ***Trans***$^{\mathsf{CGNF}}$ *process to a clique guarded formula transforms it into a set of* $\mathsf{LG}^{-}_{\approx}$ *clauses.*

Proof. This follows from **Lemma** 7.1 and **Lemma** 7.2 and the fact that the strong compatibility and the protect property hold by the applying the combination of prenex normal form and then the Skolemisation to (clique) guarded negation formulas. □

We use the notation $\mathsf{LGQ}^{-}_{\approx}$ to denote the class of the $\mathsf{LG}^{-}_{\approx}$ and $\mathsf{Q}_{\approx}$ clauses. As the class of $\mathsf{LG}^{-}_{\approx}$ clauses subsumes that of aligned guarded clauses, we put our focus on $\mathsf{LG}^{-}_{\approx}$ clauses.

Theorem 7.7. *The* ***Q-Ans***$^{\mathsf{CGNF}}$ *procedure decides satisfiability of the* $\mathsf{LGQ}^{-}_{\approx}$ *clausal class.*

Proof. By the fact that the loose guarded in the derived clauses are inherited from at least one of its premises, the protect property holds in the derived clauses. Then by **Theorem** 6.6 and **Theorem** 7.6, the statement holds. □

Next we consider back-translating $\mathsf{LGQ}^{-}_{\approx}$ clausal sets. Unlike the back-translation procedure for the class of $\mathsf{LGQ}_{\approx}$ clausal sets, applying the **Q-Abs** and the **Q-Rena** procedures to $\mathsf{LGQ}^{-}_{\approx}$ clausal sets produces $\mathsf{LGQ}^{-}_{\approx}$ *clausal sets*. The protect property ensures that in a compound-term $\mathsf{LGQ}^{-}_{\approx}$ clause C, the variables (or constants) in compound terms of C have their respective 'copy' in the (loose) guard of C, so that the derived clauses remain (loosely) guarded.

Lemma 7.16. *Suppose N is an* $\mathsf{LGQ}^{-}_{\approx}$ *clausal set. Then, the following condition hold.*

1. *N is a locally linear and locally compatible clausal set.*
2. *Applying the* ***Q-Abs*** *procedure to N transforms N to a normal, unique, locally linear and locally compatible* $\mathsf{LGQ}^{-}_{\approx}$ *clausal set N_1.*
3. *Applying the* ***Q-Rena*** *procedure to N_1 transforms N_1 to a normal, unique, globally linear and globally compatible* $\mathsf{LGQ}^{-}_{\approx}$ *clausal set N_2.*

Proof. By the protect property and **Lemma** 6.16. □

Recall that we partition a normal, unique, globally linear and globally compatible $\mathsf{LGQ}^-_\approx$ clausal set into two types of clausal sets: one is $\mathsf{LGQ}^-_\approx$ clausal sets N_1 containing only flat clauses, and another is an inter-connected compound-term $\mathsf{LGQ}^-_\approx$ clausal sets N_2. By the **Q-Unsko** procedure N_1 and N_2 are transformed into first-order formulas as they both satisfy pre-conditions for the back-translation. Moreover by the fact that N_1 consists of flat $\mathsf{LGQ}^-_\approx$ clauses, N_1 is back-translated to a (clique) guarded negation formula since each clause in N_1 is ensured to have a loose guard. Now we give the procedure so that the unskolemisation result of N_2 can be presented as a (clique) guarded negation formula. We use **D-Trans** to denote this procedure. Consider the inter-connected compound-term $\mathsf{LGQ}^-_\approx$ clausal set

$$N = \left\{ \begin{array}{l} \neg G_1(x,y) \lor A_1(f(x,y),x), \\ \neg G_2(x,y) \lor A_2(f(x,y),x), \end{array} \right\}.$$

By applying the **Q-Unsko** procedure to N, one obtains

$$F = \forall xy \exists x' \left[\begin{array}{ll} (\neg G_1(x,y) \lor A_1(x',x)) & \land \\ (\neg G_2(x,y) \lor A_2(x',x)) & \land \end{array} \right].$$

We aim to move $\exists x'$ to its quantified formulas while ensuring that subformulas in F are loosely guarded. The **D-Trans** process is given as below.

1. The first step transforms F to *disjunctive normal form*, obtaining

$$F_1 = \forall xy \exists x' \left[\begin{array}{ll} (\neg G_2(x,y) \land A_1(x',x)) & \lor \\ (\neg G_1(x,y) \land A_2(x',x)) & \lor \\ (A_1(x',x) \land A_2(x',x)) & \lor \\ (\neg G_1(x,y) \land \neg G_2(x,y)) & \end{array} \right].$$

2. Next applying the **Miniscoping** rule to F_1, moving its existential quantifications $\exists x'$ inwards as much as possible, obtaining

$$F_2 = \forall xy \left[\begin{array}{ll} (\neg G_2(x,y) \land \exists x' A_1(x',x)) & \lor \\ (\neg G_1(x,y) \land \exists x' A_2(x',x)) & \lor \\ \exists x'(A_1(x',x) \land A_2(x',x)) & \lor \\ (\neg G_1(x,y) \land \neg G_2(x,y)) & \end{array} \right].$$

3. Applying the **CNF** rules to F_2 such that distributing the (loose) guard $\neg G_1(x, y)$ and $\neg G_2(x, y)$ in the subformula $\neg G_1(x, y) \wedge \neg G_2(x, y)$ to each rest of formulas in F_2, obtaining a (clique) guarded negation formula F_3. In the immediately subformulas of F_3, either $G_1(x, y)$ or $G_2(x, y)$ is the (clique) guard. The output of this step is omitted.

Suppose N is a normal, unique, globally linear and globally compatible inter-connected $\mathsf{LGQ}^{-}_{\approx}$ clausal set consisting of clauses $C_1, \ldots, C_n$. By the protect property, each C_i in $C_1, \ldots, C_n$ contains a (loose) guard $\neg G_i$. By the fact that N is strongly and globally compatible and inter-connected, for all compound terms t in N, $\mathrm{var}(t) \subseteq \mathrm{var}(G_i)$. Hence, $\neg G_i$ can be used as a guard for any of clauses in N. Now suppose F is obtained by applying the **Q-Unsko** procedure to N. By applying Steps 1.–2. of the **D-Trans** process to F, F is reformulated as $\exists \overline{x} \forall \overline{y}(F_1 \vee \ldots \vee F_m)$. There exists F_j in $F_1, \ldots, F_m$ such that F_j is a conjunction $\neg G_1 \wedge \ldots \wedge \neg G_n$ where $\neg G_1, \ldots, \neg G_n$ are (loose) guards for $C_1, \ldots, C_n$, respectively. Then in Steps 3., by distributing the loose guards in F_j to each subformulas in F, each subformulas in F are (loosely) guarded. Hence, F can be presented as a (clique) guarded negation formula. By the above discussion, we claim:

Lemma 7.17. *Suppose F is the first-order formula obtained by applying the* ***Q-Unsko*** *procedure to a normal, unique, globally linear and globally compatible* $\mathsf{LGQ}^{-}_{\approx}$ *clausal set N. Then, applying the* ***D-Trans*** *process to F transforms it to a (clique) guarded negation formula.*

We use the notation $\textbf{Q-Rew}^{\mathsf{CGNF}}$ to denote the procedure of the saturation-based rewriting for BCQs in GNF and/or CGNF. Given a union q of BCQs, a set Σ of formulas in GNF and/or CGNF, the $\textbf{Q-Rew}^{\mathsf{CGNF}}$ procedure back-translates $\Sigma \cup \{\neg q\}$ by the following steps.

1. Apply the $\textbf{Q-Ans}^{\mathsf{CGNF}}$ procedure to $\Sigma \cup \{\neg q\}$, producing an $\mathsf{LGQ}^{-}_{\approx}$ clausal set N.

2. Apply the **Q-Abs** procedure to N, obtaining a normal, unique and strongly compatible $\mathsf{LGQ}^{-}_{\approx}$ clausal set N_1.

3. Apply the **Q-Rena** procedure to N_1, obtaining a normal, unique, globally linear and globally compatible $\mathsf{LGQ}^{-}_{\approx}$ clausal set N_2.

4. Apply the **Q-Unsko** procedure to N_2, obtaining a first-order formula F.

5. Apply the **D-Trans** process to F, obtaining a (clique) guarded negation formula F_1.

6. Negate F_1, obtaining Σ_q.

Now we give a positive answer to **Problem 9**, as the second main contribution of this chapter.

Theorem 7.8. *Suppose Σ is a set of formulas in UNF, GNF and/or CGNF, D is a set of ground atoms and q is a union of BCQs. The **Q-Rew$^{\mathsf{CGNF}}$** procedure negates the back-translation of the saturated clausal set of $\Sigma \cup \{\neg q\}$, obtaining a (clique) guarded negation formula Σ_q such that $\Sigma \cup \mathrm{D} \models q$ if and only if $\mathrm{D} \models \Sigma_q$.*

Proof. By **Theorem 5.4** and **Lemmas 7.16–7.17**. $\square$

Chapter 8

Related work

Resolution-based decision procedures

The **P-Res** rule is inspired by the 'partial replacement' strategy in [BG97, BG01] and the 'partial conclusion' of the 'Ordered Hyper-Resolution with Selection' rule in [GdN99]. The idea of 'partial conclusion' is given in [GdN99]. Without formal proofs and discussions on the integration of the 'partial conclusion' and the resolution framework of [BG01], [GdN99] claims that its result can be easily generalised into the framework of [BG01]. In [BG97, BG01] the 'partial replacement' strategy seems to be the idea behind the 'partial conclusions'. [BG97, BG01] give formal proofs to show that the 'partial replacement' strategy makes the computation of a selection-based resolution rule (the **Res** rule) redundant. However [BG97, BG01, GdN99] do not consider the 'partial replacement' strategy as a general resolution rule in the resolution framework of [BG01]. This thesis considers the **P-Res** rule as a core rule for any resolution system following the framework of [BG01]. We show that this **P-Res** rule makes a resolution inference step flexible, as one can choose any subset of the given side premises with respect to a computation of the **Res** rule. Moreover we give detailed explanations and examples to demonstrate applications of the **P-Res** rule, and formally prove that the **P-Res** rule can be used as a core rule to replace the resolution rules (with the refinement of admissible orderings and selection functions) in the resolution framework of [BG01].

Inspired by the 'MAXVAR' technique in [dNdR03] we devise the top-variable technique. The 'MAXVAR' technique and the top-variable technique are also used in [GdN99] and [ZS20a], respectively. Although [GdN99] gives a detailed example to demonstrate how the 'MAXVAR' technique is applied, it does not give formal procedures to compute the 'MAXVAR' values, formal proofs, and how the 'MAXVAR' technique is integrated into its inference system, instead [GdN99] refers readers to the manuscript of [dNdR03] for details. [dNdR03] uses the 'MAXVAR' technique to avoid term depth increase in the resolvents of loosely guarded clauses with nested compound terms allowed. In [dNdR03] the 'MAXVAR' technique is complicated: one first identifies the depth of a sequence of variable, and then applies a specially devised unification algorithm to find the 'MAXVAR'. Furthermore the 'MAXVAR' technique requires the use of non-liftable orderings, which are not compatible with the framework of [BG01] (the reasons for using the framework of [BG01] are given in the next paragraph). As a variation of the 'MAXVAR' technique, the top-variable technique, devised in [ZS20a], simplifies the procedure of computing top variables as loosely guarded clauses without nested compound terms are considered. In particular [ZS20a] generalises the top-variable technique to include query clauses. This top-variable technique uses liftable orderings, so that it fits into the framework of [BG01]. However in [ZS20a] the pre-conditions of the top-variable technique, the so-called query pair, cannot be immediately used in our query answering setting. Improving on [ZS20a, GdN99, dNdR03], this thesis gives an innovative and simple approach, namely the CompT function, to compute top variables, and encodes the top-variable technique in the plain PResT function. We formally prove that the top-variable resolution rule can be used in any saturation-based inference system following the framework of [BG01, BG98]. Moreover we generalise the premises of the top-variable resolution rule to flat clauses and (loosely) guarded clauses (with equality), with detailed proofs; see **Lemmas 4.13, 6.5** and **7.9**.

The **T-Ref$^{\mathsf{LGQ}}$** refinement extends the resolution refinement for the guarded fragment in [dNdR03, Kaz06, GdN99] and for the loosely guarded fragment in [dNdR03, GdN99, ZS20a]. Though [Kaz06] does not consider the loosely guarded fragment, it points out that by its clausification process, the obtained guarded clauses are *strongly compatible*, which is an essential property in our saturation-based rewriting procedures. Nonetheless in [Kaz06] the compatibility property is used in analysing complexity of its resolution decision procedure for the guarded fragment. [GdN99] discusses a refinement for the loosely guarded fragment, but does not give a formal description of the refinement or proofs. A detailed refinement for the loosely guarded fragment is given in [dNdR03] with formal proofs, however [dNdR03] uses non-liftable orderings, which are not compatible with the framework of [BG01]. The framework of [BG01] provides powerful simplification rules and redundancy elimination techniques, and forms the basis of the most state-of-the-art first-order theorem provers, such as Spass [WDF+09], Vampire [RV01b] and E [Sch13]. [ZS20a] only focuses on BCQ answering for the Horn fragment of LGF. This thesis devises a simple refinement (for examples **Algorithms 1** and **14**) for the whole of GF and LGF, extended to handle also BCQs. All these results are reported with detailed formal proofs.

Overall, we significantly improve and extend previous resolution decision procedures for GF and LGF in [dNdR03, Kaz06, GdN99, ZS20a]. Most importantly based on these improved resolution decision procedure, we devise the first practical BCQ answering and saturation-based BCQ rewriting procedures for whole of GF and LGF.

BCQ answering problem

Existing works consider the BCQ answering problem for Datalog$^{\pm}$ [CGL09] and description logics, such as guarded Datalog$^{\pm}$ rules [CGP15, CGL12, CGK13] and fragments of the description logic $\mathcal{ALCHOI}$ [KKZ12, CDGL+07, MRC14, RA10], respectively.

In knowledge bases a general ontological language is Datalog$^{\pm}$ rules, therefore devising automated querying procedures for Datalog$^{\pm}$ is an important task. A Datalog$^{\pm}$ rule is a first-order formula in the form

$$F = \forall \overline{x}\overline{y}(\phi(\overline{x}, \overline{y}) \rightarrow \exists \overline{z}\psi(\overline{x}, \overline{z})),$$

where $\phi(\overline{x}, \overline{y})$ and $\psi(\overline{x}, \overline{z})$ are conjunctions of atoms. Although answering BCQs for Datalog$^{\pm}$ rules is undecidable [BV81], answering BCQs for the guarded fragment of Datalog$^{\pm}$, i.e., guarded Datalog$^{\pm}$ rules, is 2ExpTime-complete [CGK13]. The above Datalog$^{\pm}$ rule F is a guarded Datalog$^{\pm}$ rule if there exists an atom in $\phi(\overline{x}, \overline{y})$ that contains all free variables of $\exists \overline{z}\psi(\overline{x}, \overline{z})$. By adopting the definition of the loosely guarded and the clique guarded fragments to Datalog$^{\pm}$ rules, guarded Datalog$^{\pm}$ can be extended to loosely guarded Datalog$^{\pm}$ and clique guarded Datalog$^{\pm}$, respectively. For example,

$$\forall xyz(\mathsf{Siblings}(x,y) \wedge \mathsf{Siblings}(y,z) \wedge \mathsf{Siblings}(z,x) \rightarrow \exists u(\mathsf{Mother}(u,x,y,z)))$$

is a loosely guarded Datalog$^{\pm}$ rule. Guarded, loosely guarded and clique guarded Datalog$^{\pm}$ can be seen as Horn fragments of GF, LGF and CGF, respectively. Hence, our query answering procedures are also the first practical decision procedures of answering BCQs for these Datalog$^{\pm}$ rules, thus providing an alternative BCQ answering procedure to traditional query answering techniques such as the chase algorithm. Note that there are guarded Datalog$^{\pm}$ rules that are not expressible in the guarded fragment (see an example in [BBtC13, Page 103]), however our **Trans**$^{\mathsf{GF}}$ process can be seen to transform guarded Datalog$^{\pm}$ rules into Horn guarded clauses.

Expressive description logic $\mathcal{ALCHOI}$ and it fragments [BHLS17] are prominent ontological languages in semantic web [Har08]. Query answering approaches for fragments of $\mathcal{ALCHOI}$ have been extensively studied in the literature; see [KKZ12, CDGL^{+}07, MRC14, RA10, Gli07]. In querying answering problems one of the key target is transforming a BCQ into knowledge bases; see the rolling-up technique [Tes01] and the tuple graph technique [CDGL98]. Interestingly the **Q-Sep** procedure also encodes query clauses to the knowledge base. By the standard translation, problems in the description logic $\mathcal{ALCHOI}$ can be translated into guarded formulas (with equality) using unary and binary predicate symbols. Thus our query answering procedures can also be used as a practical decision procedure for the BCQ answering for the description logic $\mathcal{ALCHOI}$ and its fragments.

The *squid decomposition technique* is a useful technique to analyse the complexity for answering BCQs over weakly guarded Datalog$^{\pm}$ [CGK13]. In squid decompositions, a BCQ 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, and then use ground unit resolution to eliminate the heads. In contrast, our approach first uses the separation rules to cut all 'tentacles', and then uses the top-variable resolution rule to resolve cycles in 'heads'. Our approach produces compact saturations of BCQs and the targeted guarded fragments which avoids the significant overhead of grounding, thus yielding a more practical BCQ answering procedure.

Saturation-based BCQ rewriting problem

Traditional BCQ rewriting settings consider the following problem: given a union q of BCQs, a set Σ of ontological languages, and datasets D, can we produce (function-free) first-order formulas (or Datalog-like rules) Σ_q, so that the problem of the entailment checking $D \cup \Sigma \models q$ is reduced to that of the model checking of $D \models \Sigma_q$. We say that Σ and q ensure the *first-order or* (*Datalog*) *rewritability* if Σ_q can be expressed in (function-free) first-order formulas (or Datalog rules) [CDGL+07]. Problems on the first-order (and Datalog) rewritability property has been extensively studied for fragments of the description logic [CDGL+07, HLPW18, BdBF+10, TW20, TSCS15], and for fragments of Datalog$^{\pm}$ rules [GOP14, CGL12, HLPW18, BBLP18]. However it is known that BCQ and GF (and its extensions) are not first-order or Datalog rewritable. Another interesting saturation-based rewriting approach is [HMS07], in which one first saturates axioms of the description logic $\mathcal{SHIQ}$, presenting the saturation as disjunctive Datalog, and then these disjunctive Datalog are handled by techniques of deductive databases . Unlike the *first-order* (*or Datalog*) *rewritability* and the procedure in [HMS07], our saturation-based BCQ rewriting procedure focus on back-translating clausal sets to a first-order formula, thus our rewriting procedures and existing rewriting procedures are incomparable.

Chapter 9

Conclusions

By now we have presented the *first practical* (*saturation-based*) *decision procedures* for arguably the most advanced first-order decision problems: the *Boolean conjunctive query* answering problems for the *guarded*, the *loosely guarded*, the *clique guarded*, the *unary negation*, the *guarded negation*, and the *clique guarded negation fragments*, making development of *automated decision procedures* catch up with the hunt of decidable fragments (problems) in first-order logic. Along with the developed query answering procedures, we have provided new *saturation-based Boolean conjunctive query rewriting procedures* for the considered guarded fragments. We use saturation to compile the schema and query into a first-order formula and reduce the problem to entailment checking relative to data.

Using the developed decision procedures, the research questions posed in **Problems** 4–9 have been positively answered. To start with, **Chapter** 4 develops the *resolution-based* ***P-Res*** *and top-variable inference systems*, and a *query handling procedure*, solving the BCQ answering problem for the *guarded fragment*. For the problem of query answering in the *loosely* and the *clique guarded fragments*, **Chapter** 6 devises *novel clausification processes* so that these fragments are clausified to a unified form (viz. the class of *loosely guarded clauses*), and then revises the previous *top-variable inference system* and *query handling procedures* to tackle the *loose guards*. Finally, **Chapter** 7 solves the query answering problem in the *guarded negation* and the *clique guarded negation fragments*. New clausification processes are developed to transform these fragments to the class of *loosely guarded clauses with equality*. As this clausal class allows equality, we devise the *superposition-based* ***P-Res*** *and top-variable inference systems* and redevelop the previous *query handling procedures* to accommodate equality, solving the query answering problem for the guarded negation fragments.

Another line of this thesis is the development of *saturation-based* BCQ *rewriting procedures*. Initially **Chapter 5** identifies a refined clausal class (viz. *aligned guarded clauses*) obtained by applying our clausifications to the *guarded fragment*, formally proves that the query answering procedure for the guarded fragment provides a decision procedure for this new clausal class, and gives a novel procedure for back-translating clausal sets in this class to a *first-order formula*. **Chapter 6** successfully generalises these results to the *loosely guarded* and the *clique guarded fragments*. **Chapter 7** further strengthens the previous results by sharpening the definition of *aligned clauses* and adding additional transformations, so that the back-translated first-order formula can be expressed as a (*clique*) *guarded negation formula*.

By the devised practical decision procedures, this thesis gives the following contributions.

- Our query answering procedures fill the gap of the absence of *resolution-based* (or *superposition-based*) *decision procedures* for the *clique guarded fragment* (*with equality*), and *practical decision procedures* for the *unary negation*, the *guarded negation* and the *clique guarded negation fragments*.

- Our query answering procedures provide practical solutions to real-world problems. As far as we know, our querying procedures provide the *first practical decision procedures* for the *loosely guarded*, the *clique guarded* and the *frontier guarded Datalog*$^{\pm}$ *rules* and the *first practical decision procedures for conjunctive query answering* in the *guarded*, the *loosely guarded*, the *clique guarded* and the *frontier guarded Datalog*$^{\pm}$ *rules*. Our query answering procedures also give *alternative practical decision procedures for conjunctive query answering in the expressive description logic* $\mathcal{ALCHOI}$ *and its fragments*.

- Our query answering procedures provide the theoretical foundations for *saturation-based ontology-enriched querying in any of the considered guarded fragments*.

- We have devised a series of the novel, robust and modular ***P-Res*** and *top-variable inference systems* and introduced several new *automated reasoning techniques*. These inference systems and techniques provide a powerful a toolkit for *developing practical decision procedures* for *satisfiability checking*, *conjunctive queries answering* and *back-translating tasks* for other first-order fragments.

- Our query answering procedures provide the minimal essentials for *tuning saturation-based theorem provers as preliminary querying engines*, particularly for the considered guarded fragments, thus bridging the gap of the lack of theoretical foundations for saturation-based querying.

- Our saturation-based query rewriting procedures are well-suited to provide better explanations of saturation. Our rewriting procedures allow users to view saturating clausal sets in the form of first-order formulas, thus giving users explicit information (compared to clauses) on how inferences are computed on the given queries and formulas. Moreover our approach have the flexibility that they can be combined with other reasoning methods applied to formulas, obtained by the back-translation.

Future work

Implementation We are confident that our systems provide a solid foundation for practical implementations of decision procedures and query answering systems for the family of guarded fragment considered. As we use the resolution and superposition-based framework in [BG01, BG98], any state-of-the-art saturation-based theorem prover could provide a platform for an implementation of our procedures. The key novel techniques in this thesis are *the separation rules for query clauses*, *the* ***P-Res*** *and the top-variable inference systems*, and *the rules in the back-translation procedures*.

Given a query clause Q, the application of our *separation rules* to Q consists of the following three steps:

1. Finding *surface literals* with respect to Q. Considering literals L in Q as multisets containing the variable arguments of L, one needs to implement a *multiset ordering* for literals in Q, in which the maximal multisets map to the surface literals with respect to Q.

2. Finding separable subclauses in Q. This requires us to check every pair of surface literals with respect to Q to see if they satisfy the conditions for the separation rules.

3. Separating subclauses C_1 and C_2 of Q. This can be implemented as a form of *structural transformation* with the newly introduced predicate symbols containing only the *overlapping variables* of C_1 and C_2.

Suppose the **P-Res** rule is applied to a main premise C with literals $L_1, \ldots, L_n$ selected and side premises $C_1, \ldots, C_n$. The **P-Res** resolvent of C and $C_1, \ldots, C_n$ can be computed by the following steps.

1. Without deriving resolvents apply the *selection-based resolution rule* (the ***Res** rule*) to C and $C_1, \ldots, C_n$, computing an mgu σ' for C and $C_1, \ldots, C_n$.

2. Unselect the literals $L_1, \ldots, L_n$ in C, and then select a subset $L_1, \ldots, L_m$ of $L_1, \ldots, L_n$ with $m \leq n$, performing the ***Res** rule* on $C_1, \ldots, C_m$ and C with $L_1, \ldots, L_m$ selected.

   In the application of the *top-variable resolution rule*, $L_1, \ldots, L_m$ are the *top-variable literals*, computed by the *variable ordering* with respect to σ'.

3. When the **P-Res** resolvent is derived, unselect $L_1, \ldots, L_m$.

The main techniques in our back-translation procedures are variations of the *term abstraction*, the *variable renaming* and the *unskolemisation rules*. These rules are standard rules in eliminating second-order quantifications, as implemented in the SCAN system [Ohl96]. This provides evidence that implementing our back-translation procedures is highly feasible.

Practical decision procedures for other problems As the **P-Res** and the top-variable systems are formally proved sound and refutationally complete, our inference systems are widely applicable to other problems in first-order logic (with equality). It is interesting to exploit the capability of these systems.

It is interesting to push the application of the **P-Res** rule further. For example can we use variations of the **P-Res** rule to handle transitivity relations? Particularly one needs to consider how to avoid the increasing number of distinct variables in the conclusions. Deciding the *guarded fragment with transitive guards* can be a good starting point. Although it is known that resolution decides the *guarded fragment with transitive guards* [Kaz06, KdN04], it is still interesting to see how our techniques tackle transitivity relations, since handling of transitivity opens the door for deciding and/or querying a new range of fragments such as the *expressive description logic* $\mathcal{SHI}$, the *modal logics* $K4$, $S4$ and

$S5$ and the *monadic guarded two-variable fragment with transitive guards* [GPT13]. Other possible fragments are the *triguarded fragment* [RS18] and the *guarded two-variable fragment with counting quantifiers* [Pra07].

Applications Other future work includes implementing and evaluating our query answering and rewriting procedures on real-world applications such as the *description logic* $\mathcal{ALCHOI}$ and the (*frontier*) *guarded Datalog*$^{\pm}$ *rules*, since the number of guarded formulas in the first-order theorem proving benchmark, the TPTP library [Sut16], is rather small.

Generally one reduces the problem of answering conjunctive queries to that of answering Boolean conjunctive queries. It would be interesting to investigate whether our query answering procedures can be adapted to *retrieve non-Boolean answers* from databases and knowledge bases.

We envisage that our back-translation methods could benefit the development of procedures for computing Craig interpolation and uniform interpolation (when they exist) for guarded negation fragments, but this will of course need to be investigated.

Final remark Overall, this thesis develops practical decision procedures for the conjunctive query answering and rewriting problems in a family of the guarded first-order fragments. The developed inference systems and automated reasoning techniques provide the basis for *potential practical reasoning tasks in first-order logic* (*with equality*). These procedures also lay the theoretical foundations for the possibility of developing alternative methods to traditional database approaches, based on first-order theorem proving methods.

Bibliography

[ABU79] Alfred Vaino Aho, Catriel Beeri, and Jeffrey David Ullman. The Theory of Joins in Relational Databases. *ACM Trans. Database Syst.*, 4(3):297–314, 1979.

[Ack28] Wilhelm Ackermann. Über die Erfüllbarkeit gewisser Zählausdrücke. *Math. Annalen*, 100(1):638–649, 1928.

[AdNdR99] Carlos Areces, Hans de Nivelle, and Maarten de Rijke. Prefixed Resolution: A Resolution Method for Modal and Description Logics. In *Proc. CADE'99*, volume 1632 of *LNCS*, pages 187–201. Springer, 1999.

[AdRdN01] Carlos Areces, Maarten de Rijke, and Hans de Nivelle. Resolution in modal, description and hybrid logic. *J. Logic Comput.*, 11(5):717–736, 2001.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman, 1995.

[AMdNdR99] Carlos Areces, Christof Monz, Hans de Nivelle, and Maarten de Rijke. The guarded fragment: Ins and outs. *Essays dedicated to Johan van Benthem on the occasion of his 50th birthday*, 28:1–14, 1999.

[ANvB98] Hajnal Andréka, István Németi, and Johan van Benthem. Modal Languages and Bounded Fragments of Predicate Logic. *J. Philos. Logic*, 27(3):217–274, 1998.

[AOS18] Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Simkus.

Rewriting Guarded Existential Rules into Small Datalog Programs. In *Proc. ICDT'18*, volume 98 of *LIPIcs*, pages 4:1–4:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[AOS20] Medina Andresel, Magdalena Ortiz, and Mantas Simkus. Query Rewriting for Ontology-Mediated Conditional Answers. In *Proc. AAAI'20*, pages 2734–2741. AAAI, 2020.

[BBGP21] Pablo Barceló, Gerald Berger, Georg Gottlob, and Andreas Pieris. Guarded Ontology-Mediated Queries. In Judit Madarász and Gergely Székely, editors, *Hajnal Andréka and István Németi on Unity of Science: From Computing to Relativity Theory Through Algebraic Logic*, pages 27–52. Springer, 2021.

[BBLP18] Pablo Barceló, Gerald Berger, Carsten Lutz, and Andreas Pieris. First-Order Rewritability of Frontier-Guarded Ontology-Mediated Queries. In *Proc. IJCAI'18*, pages 1707–1713. IJCAI, 2018.

[BBtC13] Vince Bárány, Michael Benedikt, and Balder ten Cate. Rewriting Guarded Negation Queries. In *Proc. MFCS'13*, pages 98–110. Springer, 2013.

[BBtC18] Vince Bárány, Michael Benedikt, and Balder ten Cate. Some Model Theory of Guarded Negation. *J. Symb. Logic*, 83(4):1307–1344, 2018.

[BdBF+10] Alexander Borgida, Jos de Bruijn, Enrico Franconi, Inanç Seylan, Umberto Straccia, David Toman, and Grant E. Weddell. On Finding Query Rewritings under Expressive Constraints. In *Proc. SEDB'10*, pages 426–437. Esculapio Editore, 2010.

[BFMY83] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the Desirability of Acyclic Database Schemes. *J. ACM*, 30(3):479–513, 1983.

[BG90] Leo Bachmair and Harald Ganzinger. On Restrictions of Ordered Paramodulation with Simplification. In *Proc. CADE'90*, volume 449 of *LNCS*, pages 427–441. Springer, 1990.

[BG97] Leo Bachmair and Harald Ganzinger. A theory of resolution. Research Report MPI-I-97-2-005, Max-Planck-Institut für Informatik, 1997.

[BG98] Leo Bachmair and Harald Ganzinger. Equational Reasoning in Saturation-Based Theorem Proving. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction: A Basis for Applications*, pages 353–397. Kluwer, 1998.

[BG01] Leo Bachmair and Harald Ganzinger. Resolution Theorem Proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.

[BGG97] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer, 1997.

[BGO14] Vince Bárány, Georg Gottlob, and Martin Otto. Querying the Guarded Fragment. *Logic Methods Comput. Sci.*, 10(2), 2014.

[BGW93] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In *Proc. KGC'93*, volume 713 of *LNCS*, pages 83–96. Springer, 1993.

[BHLS17] Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. *An Introduction to Description Logic*. Cambridge Univ. Press, 2017.

[BKK+18] Meghyn Bienvenu, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, and Michael Zakharyaschev. Ontology-Mediated Queries: Combined Complexity and Succinctness of Rewritings via Circuit Complexity. *J. ACM*, 65(5):28:1–28:51, 2018.

[BKM+17] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the Chase. In *Proc. PODS'17*, pages 37–52. ACM, 2017.

[BLMS94] Ronen I. Brafman, Jean-Claude Latombe, Yoram Moses, and Yoav Shoham. Knowledge as a tool in motion planning under uncertainty. In Ronald Fagin, editor, *Theoretical Aspects of Reasoning About Knowledge*, pages 208–224. Morgan Kaufmann, 1994.

[BLMS11] Jean-François Baget, Michel Leclére, Marie-Laure Mugnier, and Eric Salvat. On Rules with Existential Variables: Walking the Decidability Line. *Artif. Intell.*, 175(9):1620–1654, 2011.

[BN07] Franz Baader and Werner Nutt. Basic Description Logics. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 47–104. Cambridge Univ. Press, 2 edition, 2007.

[BRV01a] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge Tracts in Theor. Comp. Sci. Cambridge Univ. Press, 2001.

[BRV01b] Patrick Blackburn, Maarten de Rijke, and Yde Venema. Modal Logic. Cambridge Tracts in Theor. Comp. Sci., chapter 2, pages 83–90. Cambridge Univ. Press, 2001.

[BS28] Paul Bernays and Moses Schönfinkel. Zum entscheidungsproblem der mathematischen logik. *Math. Annalen*, 99(1):342–372, 1928.

[BtCO12] Vince Bárány, Balder ten Cate, and Martin Otto. Queries with Guarded Negation. *Proc. VLDB Endow.*, 5(11):1328–1339, 2012.

[BtCS15] Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded Negation. *J. ACM*, 62(3):22:1–22:26, 2015.

[BV81] Catriel Beeri and Moshe Y. Vardi. The Implication Problem for Data Dependencies. Springer, 1981.

[BvB07] Patrick Blackburn and Johan van Benthem. Modal logic: a semantic perspective. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of

Studies in logic and practical reasoning, pages 1–84. North-Holland, 2007.

[BvBW07] Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors. *Handbook of Modal Logic*, volume 3 of *Studies in logic and practical reasoning*. North-Holland, 2007.

[CCK+17] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semant. Web*, 8(3):471–487, 2017.

[CDGL98] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the Decidability of Query Containment under Constraints. In *Proc. PODS'98*, pages 149–158, 1998.

[CDGL+07] 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.

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[CG07] Diego Calvanese and Giuseppe De Giacomo. Expressive Description Logics. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 193–236. Cambridge Univ. Press, 2 edition, 2007.

[CGK13] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the Infinite Chase: Query Answering Under Expressive Relational Constraints. *J. Artif. Intell. Res.*, 48(1):115–174, 2013.

[CGL+07] 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. Autom. Reason.*, 39(3):385–429, 2007.

[CGL09] 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.

[CGL+10] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications. In *Proc. LICS'10*, pages 228–242. IEEE, 2010.

[CGL+11] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The MASTRO system for ontology-based data access. *Semant. Web*, 2(1):43–53, 2011.

[CGL12] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A General Datalog-based Framework for Tractable Query Answering over Ontologies. *J. Web Semant.*, 14:57–83, 2012.

[CGP15] Marco Calautti, Georg Gottlob, and Andreas Pieris. Chase termination for guarded existential rules. In *Proc. PODS'15*, pages 91–103. ACM, 2015.

[CGT89] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. Knowl. Data Eng.*, 1(1):146–166, 1989.

[Chu36] Alonzo Church. A note on the Entscheidungsproblem. *J. Symb. Logic*, 1(1):40–41, 1936.

[CM77] 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.

[Cra57a] William Craig. Linear reasoning. a new form of the herbrand-gentzen theorem. *J. Symb. Logic*, 22(3):250–268, 1957.

[Cra57b] William Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symb. Logic*, 22(3):269–285, 1957.

[CRSS94] David Cyrluk, S. Rajan, Natarajan Shankar, and Mandayam K. Srivas. Effective Theorem Proving for Hardware Verification. In *Proc. TPCD'94*, volume 901 of *LNCS*, pages 203–222. Springer, 1994.

[CTS11] Alexandros Chortaras, Despoina Trivela, and Giorgos B. Stamou. Optimized Query Rewriting for OWL 2 QL. In *Proc. CADE'11*, volume 6803 of *LNCS*, pages 192–206. Springer, 2011.

[dCCF82] José Mauro Volkmer de Castilho, Marco A. Casanova, and Antonio L. Furtado. A Temporal Framework for Database Specifications. In *Proc. VLDB'82*, pages 280–291. Morgan Kaufmann, 1982.

[DFK+08] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Li Ma, Edith Schonberg, Kavitha Srinivas, and Xingzhi Sun. Scalable Grounded Conjunctive Query Evaluation over Large and Expressive Knowledge Bases. In *Proc. ISWC'08*, volume 5318 of *LNCS*, pages 403–418. Springer, 2008.

[DG79] Burton Dreben and Warren D. Goldfarb. *The Decision Problem: Solvable Classes of Quantificational Formulas*. Addison-Wesley, 1979.

[DL15] Giovanna D'Agostino and Giacomo Lenzi. Bisimulation quantifiers and uniform interpolation for guarded first order logic. *Theor. Comput. Sci.*, 563:75–85, 2015.

[dN00] Hans de Nivelle. Deciding the E^+ - class by an a posteriori, liftable order. *Ann. Pure Appl. Logic*, 104(1-3):219–232, 2000.

[dNdR03] Hans de Nivelle and Maarten de Rijke. Deciding the Guarded Fragments by Resolution. *J. Symb. Comput.*, 35(1):21–58, 2003.

[dNP01] Hans de Nivelle and Ian Pratt-Hartmann. A Resolution-Based Decision Procedure for the Two-Variable Fragment with Equality. In *Proc. IJCAR'01*, volume 2083 of *LNCS*, pages 211–225. Springer, 2001.

[DV01a] Anatoli Degtyarev and Andrei Voronkov. Equality Reasoning in Sequent-Based Calculi. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 611–706. Elsevier and MIT Press, 2001.

[DV01b] Anatoli Degtyarev and Andrei Voronkov. The Inverse Method. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 179–272. Elsevier and MIT Press, 2001.

[Eng96] Thorsten Engel. Quantifier Elimination in Second-Order Predicate Logic. Diplomarbeit, Fachbereich Informatik, Univ. des Saarlandes, Saarbrücken, Germany, October 1996.

[FFG02] Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

[Fit00] Melvin Fitting. Modality and Databases. In *Proc. TABLEAUX'00*, volume 1847 of *LNCS*, pages 19–39. Springer, 2000.

[FKL19] Cristina Feier, Antti Kuusisto, and Carsten Lutz. Rewritability in Monadic Disjunctive Datalog, MMSNP, and Expressive Description Logics. *Logic Methods Comput. Sci.*, 15(2), 2019.

[FLHT01] 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*, pages 1791–1849. Elsevier and MIT Press, 2001.

[FLTZ93] Christian G. Fermüller, Alexander Leitsch, Tanel Tammet, and N. K. Zamov. *Resolution Methods for the Decision Problem*, volume 679 of *LNCS*. Springer, 1993.

[FN71] Richard Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.*, 2(3/4):189–208, 1971.

[Gab81] Dov M. Gabbay. *Expressive Functional Completeness in Tense Logic (Preliminary report)*, pages 91–117. Springer, 1981.

[GdN99] 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.

[GHMS98] 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.

[GHS02] Lilia Georgieva, Ullrich Hustadt, and Renate A. Schmidt. A New Clausal Class Decidable by Hyperresolution. In *Proc. CADE'02*, volume 2392 of *LNCS*, pages 260–274. Springer, 2002.

[GHS03] Lilia Georgieva, Ullrich Hustadt, and Renate A. Schmidt. Hyperresolution for guarded formulae. *J. Symb. Comput.*, 36(1-2):163–192, 2003.

[GKV97] Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the Decision Problem for Two-Variable First-Order Logic. *The Bulletin of Symb. Logic*, 3(1):53–69, 1997.

[GL75] Warren D. Goldfarb and Harry R. Lewis. Skolem Reduction Classes. *J. Symb. Logic*, 40(1):62–68, 1975.

[Gli07] Birte Glimm. *Querying Description Logic Knowledge Bases*. PhD thesis, Univ. Manchester, Manchester, U.K., 2007.

[GLS03] 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.

[GMV99] Harald Ganzinger, Christoph Meyer, and Margus Veanes. The Two-Variable Guarded Fragment with Transitive Relations. In *Proc. LICS'99*, pages 24–34. IEEE, 1999.

[GO99] Erich Grädel and Martin Otto. On Logics with Two Variables. *Theor. Comput. Sci.*, 224(1-2):73–113, 1999.

[Göd30] Kurt Gödel. Die vollständigkeit der axiome des logischen funktionenkalküls. *Monatshefte für Mathematik und Physik*, 37(1):349–360, 1930.

[Göd31] Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931.

[Göd32] Kurt Gödel. Ein Spezialfall des Entscheidungsproblems der theoretischen Logik. *Ergebnisse eines mathematischen Kolloquiums*, 2:27–28, 1932.

[Gol84] Warren D. Goldfarb. The Unsolvability of the Godel Class with Identity. *J. Symb. Logic*, 49(4):1237–1252, 1984.

[GOP14] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Query Rewriting and Optimization for Ontological Databases. *ACM Trans. Database Syst.*, 39(3):25:1–25:46, 2014.

[GOR99] Erich Grädel, Martin Otto, and Eric Rosen. Undecidability results on two-variable logics. *Arch. Math. Logic*, 38(4-5):313–354, 1999.

[GPT13] Georg Gottlob, Andreas Pieris, and Lidia Tendera. Querying the Guarded Fragment with Transitivity. In *Proc. ICALP'13*, volume 7966 of *LNCS*, pages 287–298. Springer, 2013.

[GR69] Larry Wos George Robinson. Paramodulation and Theorem-proving in First-Order Theories with Equality. *Machine intelligence*, 4:135–150, 1969.

[Grä99a] Erich Grädel. Decision Procedures for Guarded Logics. In *Proc. CADE'16*, volume 1632 of *LNCS*, pages 31–51. Springer, 1999.

[Grä99b] Erich Grädel. On the Restraining Power of Guards. *J. Symb. Logic*, 64(4):1719–1742, 1999.

[Grä03] Erich Grädel. Decidable fragments of first-order and fixed-point logic – From prefix-vocabulary classes to guarded logics. In *Proc. Kalmár Workshop on Logic and Comput. Sci.*, 2003.

[Gre69] C. Cordell Green. Application of Theorem Proving to Problem Solving. In Donald E. Walker and Lewis M. Norton, editors, *Proc. IJCAI'69*, pages 219–240. William Kaufmann, 1969.

[GRS14] Georg Gottlob, Sebastian Rudolph, and Mantas Simkus. Expressiveness of guarded existential rule languages. In *Proc. PODS'14*, pages 27–38. ACM, 2014.

[GSS08a] Dov M. Gabbay, Renate A. Schmidt, and Andrzej Szałas. *Second-order Quantifier Elimination*. College publications, 2008.

[GSS08b] Dov M. Gabbay, Renate A. Schmidt, and Andrzej Szałas. *Second-order Quantifier Elimination*, chapter 5, pages 63–69. College publications, 2008.

[Gur65] Yuri Gurevich. Existential interpretation. *Algebra and logic*, 4(4):71–84, 1965.

[Gur68] Yuri Gurevich. *The Decision Problem for Some Algebraic Theories*. PhD thesis, Ural State Univ., Sverdlovsk, USSR, 1968.

[HA28] David Hilbert and Wilhelm Ackermann. *Grundzüge der theoretischen Logik*. Springer, 1928.

[Häh01] Reiner Hähnle. Tableaux and Related Methods. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 100–178. Elsevier and MIT Press, 2001.

[Har08] John Harrison. Theorem Proving for Verification (Invited Tutorial). In *Proc. CAV'08*, volume 5123 of *LNCS*, pages 11–18. Springer, 2008.

[HdNS00] Ullrich Hustadt, Hans de Nivelle, and Renate A. Schmidt. Resolution-Based Methods for Modal Logics. *Logic J. IGPL*, 8(3):265–292, 2000.

[Hen63] Leon Henkin. An Extension of the Craig-Lyndon Interpolation Theorem. *J. Symb. Logic*, 28(3):201–216, 1963.

[Her31] Jacques Herbrand. Sur le problème fondamental de la logique mathématique. *Comptes Rendus Soc. Sci. Lett. Varsovie, Classe III*, 24:12–56, 1931.

[HF89] Joseph Y. Halpern and Ronald Fagin. Modelling Knowledge and Action in Distributed Systems. *Distributed Comput.*, 3(4):159–177, 1989.

[Hla02] 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.

[HLPW18] André Hernich, Carsten Lutz, Fabio Papacchini, and Frank Wolter. Horn-Rewritability vs PTime Query Evaluation in Ontology-Mediated Querying. In *Proc. IJCAI'18*, pages 1861–1867, 2018.

[HM02] Eva Hoogland and Maarten Marx. Interpolation and Definability in Guarded Fragments. *Studia Logica*, 70(3):373–409, 2002.

[HMA$^+$08] Stijn Heymans, Li Ma, Darko Anicic, Zhilei Ma, Nathalie Steinmetz, Yue Pan, Jing Mei, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Edith Schonberg, Kavitha Srinivas, Cristina Feier, Graham Hench, Branimir Wetzstein, and Uwe Keller. Ontology Reasoning with Large Data Repositories. In Martin Hepp, Pieter De Leenheer, Aldo de Moor, and York Sure, editors, *Ontology Management, Semantic Web, Semantic Web Services, and Business Applications*, volume 7 of *Semantic Web and Beyond: Computing for Human Experience*, pages 89–128. Springer, 2008.

[HMS07] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *J. Autom. Reason.*, 39(3):351–384, 2007.

[HMS08] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Deciding expressive description logics in the framework of resolution. *Inf. Comput.*, 206(5):579–601, 2008.

[Hod02] Ian Hodkinson. Loosely Guarded Fragment of First-Order Logic Has the Finite Model Property. *Studia Logica*, 70(2):205–240, 2002.

[HPMW07] Ian Horrocks, Peter F. Patel-Schneider, Deborah L. McGuinness, and Christopher A. Welty. OWL: a Description-Logic-Based Ontology Language for the Semantic Web. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 458–486. Cambridge Univ. Press, 2 edition, 2007.

[HPvH03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *J. Web Semant.*, 1(1):7–26, 2003.

[HS99] Ullrich Hustadt and Renate A. Schmidt. Maslov's Class K Revisited. In *Proc. CADE'99*, volume 1632 of *LNCS*, pages 172–186. Springer, 1999.

[HT01] Colin Hirsch and Stephan Tobies. A Tableau Algorithm for the Clique Guarded Fragment. In *Advances in Modal Logics Volume 3*. CSLI, 2001.

[Hus99] Ullrich Hustadt. *Resolution Based Decision Procedures for Subclasses of First-order Logic*. PhD thesis, Univ. Saarlandes, Saarbrücken, Germany, 1999.

[Joy76] William H. Joyner. Resolution Strategies as Decision Procedures. *J. ACM*, 23(3):398–417, 1976.

[Kal33] László Kalmár. Über die Erfüllbarkeit derjenigen Zählausdrücke, welche in der Normalform zwei benachbarte Allzeichen enthalten. *Math. Annalen*, 108(1):466–484, 1933.

[Kal37] László Kalmár. Zurückführung des Entscheidungsproblems auf den Fall von Formeln mit einer einzigen, binären, Funktionsvariablen. *Compositio Mathematica*, 4:137–144, 1937.

[Kaz06] Yevgeny Kazakov. *Saturation-Based Decision Procedures for Extensions of the Guarded Fragment*. PhD thesis, Univ. Saarlandes, Saarbrücken, Germany, 2006.

[KB83] Donald Ervin Knuth and Peter Bendix. *Simple Word Problems in Universal Algebras*, pages 342–376. Springer, 1983.

[KdN04] Yevgeny Kazakov and Hans de Nivelle. A Resolution Decision Procedure for the Guarded Fragment with Transitive Guards. In *Proc. IJCAR'04*, pages 122–136. Springer, 2004.

[KKZ12] Stanislav Kikot, Roman Kontchakov, and Michael Zakharyaschev. Conjunctive query answering with OWL 2 QL. In *Proc. KR'12*, pages 275–285. AAAI, 2012.

[KM08] Yevgeny Kazakov and Boris Motik. A Resolution-Based Decision Procedure for $\mathcal{SHOIQ}$. *J. Autom. Reason.*, 40(2-3):89–116, 2008.

[KNG16] Mark Kaminski, Yavor Nenov, and Bernardo Cuenca Grau. Datalog rewritability of Disjunctive Datalog programs and non-Horn ontologies. *Artif. Intell.*, 236:90–118, 2016.

[Kog12] Mikhail R. Kogalovsky. Ontology-based data access systems. *Program. Comput. Softw.*, 38(4):167–182, 2012.

[KRZ13] Roman Kontchakov, Mariano Rodriguez-Muro, and Michael Zakharyaschev. Ontology-based data access with databases: A short course. In Sebastian Rudolph, Georg Gottlob, Ian Horrocks, and Frank van Harmelen, editors, *Proc. Reasoning Web Summer School*, volume 8067 of *LNCS*, pages 194–229. Springer, 2013.

[KV00] Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000.

[Lew79] Harry R. Lewis. *Unsolvable Classes of Quantificational Formulas*. Addison-Wesley, 1979.

[Lia03] Churn-Jung Liau. Belief, information acquisition, and trust in multi-agent systems–A modal logic formulation. *Artif. Intell.*, 149(1):31–60, 2003.

[Löw15] Leopold Löwenheim. Über möglichkeiten im relativkalkül. *Mathematische Annalen*, 76:447–470, 1915.

[LST99] Carsten Lutz, Ulrike Sattler, and Stephan Tobies. A Suggestion for an n-ary Description Logic. In *Proc. DL'99*, volume 22 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1999.

[Mar88] V. Wiktor Marek. A Natural Semantics for Modal Logic Over Databases. *Theor. Comput. Sci.*, 56:187–209, 1988.

[Mar07] Maarten Marx. Queries determined by views: Pack your views. In *Proc. PODS'07*, pages 23–30. ACM, 2007.

[MGS+19] Mohamed Nadjib Mami, Damien Graux, Simon Scerri, Hajira Jabeen, Sören Auer, and Jens Lehmann. Squerall: Virtual Ontology-Based Access to Heterogeneous and Large Data Sources. In *Proc. ISWC'19*, volume 11779 of *LNCS*, pages 229–245. Springer, 2019.

[MH69] John McCarthy and Patrick J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh Univ. Press, 1969.

[MMS79] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing Implications of Data Dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

[Moo10] J Strother Moore. Theorem Proving for Verification: The Early Days. In *Proc. LICS'10*, page 283. IEEE, 2010.

[Mor75] Michael Mortimer. On languages with two variables. *Math. Logic Q.*, 21(1):135–140, 1975.

[Mot06] Boris Motik. *Reasoning in description logics using resolution and deductive databases*. PhD thesis, Karlsr. Inst. of Technology, Germany, 2006.

[MRC14] 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.

[MW97] William McCune and Larry Wos. Otter - The CADE-13 Competition Incarnations. *J. Autom. Reason.*, 18(2):211–220, 1997.

[NDH19] Cláudia Nalon, Clare Dixon, and Ullrich Hustadt. Modal Resolution: Proofs, Layers, and Refinements. *ACM Trans. Comput. Logic*, 20(4):23:1–23:38, 2019.

[NML+19] M. Saqib Nawaz, Moin Malik, Yi Li, Meng Sun, and Muhammad Ikram Ullah Lali. A Survey on Theorem Provers in Formal Methods. *CoRR*, abs/1912.03028, 2019.

[NS56] Allen Newell and Herbert A. Simon. The logic theory machine–A complex information processing system. *IRE Trans. on Inf. Theory*, 2(3):61–79, 1956.

[NW01] Andreas Nonnengart and Christoph Weidenbach. Computing Small Clause Normal Forms. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 335–367. Elsevier and MIT Press, 2001.

[Ohl96] Hans Jürgen Ohlbach. SCAN—Elimination of predicate quantifiers. In *Proc. CADE'96*, pages 161–165. Springer, 1996.

[PCS14] Freddy Priyatna, Óscar Corcho, and Juan F. Sequeda. Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. In *Proc. WWW'14*, pages 479–490. ACM, 2014.

[PHM09] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. Efficient Query Answering for OWL 2. In *Proc. ISWC'09*, volume 5823 of *LNCS*, pages 489–504. Springer, 2009.

[PLC+08] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking Data to Ontologies. In Stefano Spaccapietra, editor, *J. on Data Semantics X*, pages 133–173. Springer, 2008.

[Pop94] Sally Popkorn. *First Steps in Modal Logic*. Cambridge Univ. Press, 1994.

[Pra80] Vaughan R. Pratt. Application of Modal Logic to Programming. *Studia Logica*, 39:257–274, 1980.

[Pra07] Ian Pratt-Hartmann. Complexity of the Guarded Two-variable Fragment with Counting Quantifiers. *J. Logic Comput.*, 17(1):133–155, 2007.

[RA10] Riccardo Rosati and Alessandro Almatelli. Improving Query Answering over DL-Lite Ontologies. In *Proc. KR'10*, pages 290–300. AAAI, 2010.

[RN20] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

[Rob65a] John Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1):23–41, 1965.

[Rob65b] John Alan Robinson. Automatic deduction with hyper-resolution. *Int. J. Comp. Math.*, 1:227–234, 1965.

[RS18] Sebastian Rudolph and Mantas Simkus. The Triguarded Fragment of First-Order Logic. In *Proc. LPAR'18*, volume 57, pages 604–619. EasyChair, 2018.

[RU12] Nicholas Rescher and Alasdair Urquhart. *Temporal logic*, volume 3. Springer Science & Business Media, 2012.

[RV01a] Alexandre Riazanov and Andrei Voronkov. Splitting without Backtracking. In *Proc. IJCAI'01*, pages 611–617. Morgan Kaufmann, 2001.

[RV01b] Alexandre Riazanov and Andrei Voronkov. Vampire 1.1 (System Description). In *Proc. IJCAR'01*, volume 2083 of *LNCS*, pages 376–380. Springer, 2001.

[Sch34] Kurt Schütte. Untersuchungen zum Entscheidungsproblem der mathematischen Logik. *Math. Annalen*, 109(1):572–603, 1934.

[Sch96] Renate A. Schmidt. Resolution is a Decision Procedure for Many Propositional Modal Logics. In *Proc. AiML'96*, pages 189–208. CSLI, 1996.

[Sch98] Renate A. Schmidt. Decidability by unrefined resolution for propositional modal logics. In *Proc. Int. Semin. RelMiCS'98*, pages 192–196, 1998.

[Sch99] Renate A. Schmidt. Decidability by Resolution for Propositional Modal Logics. *J. Autom. Reason.*, 22(4):379–396, 1999.

[Sch01] Johann Schumann. *Automated theorem proving in software engineering*. Springer, 2001.

[Sch13] Stephan Schulz. System Description: E 1.8. In *Proc. LPAR'13*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.

[Sco62] Dana Scott. A decision method for validity of sentences in two variables. *J. Symb. Logic*, 27(377):74, 1962.

[Seg82] Krister Segerberg. A completeness theorem in the modal logic of programs. *Banach Center Publications*, 9(1):31–46, 1982.

[Seg17] Luc Segoufin. A Survey on Guarded Negation. *ACM SIGLOG News*, 4(3):12–26, 2017.

[SH00] 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.

[SH13] Renate A. Schmidt and Ullrich Hustadt. First-Order Resolution Methods for Modal Logics. In *Programming Logics - Essays in Memory of Harald Ganzinger*, volume 7797 of *LNCS*, pages 345–391. Springer, 2013.

[SM13] Juan F. Sequeda and Daniel P. Miranker. Ultrawrap: SPARQL execution on relational data. *J. Web Semant.*, 22:19–39, 2013.

[ST04] Wieslaw Szwast and Lidia Tendera. The guarded fragment with transitive guards. *Ann. Pure Appl. Logic*, 128(1-3):227–276, 2004.

[Sur59] János Surányi. *Reduktionstheorie des Entscheidungsproblems im Prädikatenkalkül der ersten Stufe*. Ungarische Akademie der Wissenschaften, 1959.

[Sut] Geoff Sutcliffe. Geoff sutcliffe's overview of automated theorem proving. `http://tptp.org/OverviewOfATP.html`. Online; last accessed: 04 Nov. 2021.

[Sut16] Geoff Sutcliffe. The CADE ATP System Competition - CASC. *AI Magazine*, 37(2):99–101, 2016.

[tCS13] Balder ten Cate and Luc Segoufin. Unary negation. *Logic Methods Comput. Sci.*, 9(3), 2013.

[Tes01] Sergio Tessaris. *Questions and Answers: Reasoning and Querying in Description Logic*. PhD thesis, Univ. Manchester, Manchester, U.K., 2001.

[TSCS15] Despoina Trivela, Giorgos Stoilos, Alexandros Chortaras, and Giorgos B. Stamou. Optimising resolution-based rewriting algorithms for OWL ontologies. *J. Web Semant.*, 33:30–49, 2015.

[Tur36] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. the London Math. Soc.*, 2(42):230–265, 1936.

[TW20] David Toman and Grant E. Weddell. First Order Rewritability for Ontology Mediated Querying in Horn-$\mathcal{DLFD}$. In *Proc DL'20*, volume 2663. CEUR-WS.org, 2020.

[TW21] David Toman and Grant E. Weddell. FO Rewritability for OMQ using Beth Definability and Interpolation. In *Proc. DL'21*, volume 2954 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volumes I and II*. Comp. Sci. Press, 1989.

[Var95] Moshe Y. Vardi. On the Complexity of Bounded-Variable Queries. In Mihalis Yannakakis and Serge Abiteboul, editors, *Proc. PODS'95*, pages 266–276. ACM, 1995.

[Var96] Moshe Y. Vardi. Why is Modal Logic So Robustly Decidable? In *Proc. DIMACS Workshop'96*, pages 149–183. DIMACS/AMS, 1996.

[Var00] Moshe Y. Vardi. Constraint satisfaction and database theory: A tutorial. In *Proc. PODS'00*, pages 76–85. ACM, 2000.

[vB91] Johan van Benthem. Temporal logic. Research Report x-91-05, Institute for Logic, Language and Computation, Univ. Amsterdam, 1991.

[vB97] Johan van Benthem. Dynamic Bits and Pieces. Research Report LP-97-01, Univ. Amsterdam, 1997.

[WDF+09] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS Version 3.5. In *Proc. CADE'09*, volume 5663 of *LNCS*, pages 140–145. Springer, 2009.

[Wei01] Christoph Weidenbach. Combining Superposition, Sorts and Splitting. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1965–2013. Elsevier and MIT Press, 2001.

[WRC65] Larry Wos, George A. Robinson, and Daniel F. Carson. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *J. ACM*, 12(4):536–541, 1965.

[XCK+18] Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. Ontology-Based Data Access: A Survey. In Jérôme Lang, editor, *Proc. IJCAI'18*, pages 5511–5519. IJCAI, 2018.

[YO79] Clement Yu and Meral Ozsoyoglu. An Algorithm for Tree-query Membership of a Distributed Query. In *Proc. COMPSAC'79*, pages 306–312. IEEE, 1979.

[ZHD09] Lan Zhang, Ullrich Hustadt, and Clare Dixon. A Refined Resolution Calculus for CTL. In *Proc. CADE'09*, volume 5663 of *LNCS*, pages 245–260. Springer, 2009.

[ZS20a] 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.

[ZS20b] Sen Zheng and Renate A. Schmidt. Querying the guarded fragment via resolution (extended abstract). In *Proc. PAAR'20*, volume 2752 of *CEUR Workshop Proceedings*, pages 167–177. CEUR-WS.org, 2020.

Index

A