Queries with Guarded Negation (full version)[*]

Vince Bárány
TU Darmstadt, Germany
barany@mathematik.tu-darmstadt.de

Balder ten Cate
UC Santa Cruz, CA, USA
btencate@ucsc.edu

Martin Otto
TU Darmstadt, Germany
otto@mathematik.tu-darmstadt.de

ABSTRACT

A well-established and fundamental insight in database theory is that negation (also known as complementation) tends to make queries difficult to process and difficult to reason about. Many basic problems are decidable and admit practical algorithms in the case of unions of conjunctive queries, but become difficult or even undecidable when queries are allowed to contain negation. Inspired by recent results in finite model theory, we consider a restricted form of negation, *guarded negation*. We introduce a fragment of SQL, called GN-SQL, as well as a fragment of Datalog with stratified negation, called GN-Datalog, that allow only guarded negation, and we show that these query languages are computationally well behaved, in terms of testing query containment, query evaluation, open-world query answering, and boundedness. GN-SQL and GN-Datalog subsume a number of well known query languages and constraint languages, such as unions of conjunctive queries, monadic Datalog, and frontier-guarded tgds. In addition, an analysis of standard benchmark workloads shows that most usage of negation in SQL in practice is guarded negation.

1. INTRODUCTION

A well-established and fundamental insight of database theory is that *negation* (also called *complementation* or *difference*) tends to make queries difficult to reason about. Recall that the unions of conjunctive queries are the first-order queries that can be expressed without using negation. Many basic problems are decidable and admit practical algorithms in the case of unions of conjunctive queries, but are undecidable in the case of arbitrary first-order queries. Examples include *query containment* and *open world query answering.*

We argue that most queries in practice use only a restricted form of negation, which is called *guarded negation* and was first considered in [7] (in the study of decidable fragments of first-order logic). By guarded negation we mean that queries may involve negative conditions only if these conditions, intuitively, pertain to a single record in the database. For instance, if a database schema contains relations Author(AuthID,Name) and Book(AuthID,Title,Year,Publisher), the query that asks for *authors that did not publish any book with Elsevier* since "not publishing a book with Elsevier" is a property of an author. The query that asks for *pairs of authors and book titles where the author did not publish the book*, on the other hand, is not allowed, since it involves a negative condition (in this case, an inequality) pertaining to two values that do not necessarily co-occur in a record in the database. The requirement of guarded negation can be formally stated most easily in terms of the Relational Algebra: we allow the use of the difference operator $E_1 - E_2$ provided that E_1 is a projection of a relation from the database.

Based on an analysis of standard SQL benchmark workloads, we argue that guarded negation covers most uses of negation in SQL in practice. Furthermore, building on recent results in logic and finite model theory [7, 10], we show that queries with guarded negation are computationally very well behaved. For instance, query containment and open world query answering are decidable for first-order queries with guarded negation, and boundedness is decidable for the guarded-negation fragment of Datalog with stratified negation. We also determine the complexity of query evaluation for queries with guarded negation, which (under reasonable complexity theoretic assumptions) is easier than the same problem for queries with unguarded negation.

Our results show that guarded negation is a fruitful concept for databases, in the sense that it enables solving central decision problems in database theory more efficiently. We also believe that guarded negation is a fruitful concept from a more practical point of view, allowing for efficient query plans and query optimization strategies. This is something we are exploring in a separate line of investigation.

Outline and Main Results. In Section 2 we review the definition of GNFO, guarded-negation first-order logic, and GNFP, guarded-negation fixed-point logic, as well as the main known decidability and complexity results for these logics [7]. We also provide an equivalent characterization of GNFO in terms of the Relational Algebra.

In Section 3, we investigate what it means for an SQL query to be negation-guarded. Specifically, we identify syntactic restrictions on the use of negation in SQL queries, and we show that the first-order queries satisfying these restrictions can be translated to GNFO, and, in fact, are ex-

[*]Detailed proofs of the results in this paper can be found in the full version available at `tiny.cc/guardednegation`. We would like to thank Alkis Polyzotis for helpful comments.

pressively complete for GNFO, in the sense of Codd's completeness theorem. Furthermore, by means of an analysis of standard SQL benchmark workloads, we show that most SQL queries in practice satisfy the syntactic restrictions.

In Section 4, similarly, we introduce a syntactic fragment of Datalog with stratified negation, called GN-Datalog, which admits a translation into GNFP.

In Section 5, we show that GN-SQL and GN-Datalog subsume a number of important existing query languages and constraint languages. In particular, GN-Datalog subsumes both monadic Datalog (which it extends by allowing IDBs of arbitrary arity, and negation, subject to guardedness conditions) and unions of conjunctive queries.

In Section 6, we show that query containment is 2ExpTime-complete for GN-SQL queries as well as for GN-Datalog queries (note that the decidability of these problems follows via translations into GNFO and GNFP).

In Section 7, we determine the complexity of query evaluation and open-world query answering for GN-SQL and for GN-Datalog. While the data complexity of query evaluation is in PTime, both for GN-SQL and for GN-Datalog, in terms of combined complexity, the problem is complete for the complexity class $\mathrm{P}^{\mathrm{NP}[\log^2]}$ (for GN-SQL) and P^{NP} (for GN-Datalog). The data complexity of *open world query answering* for GN-SQL with respect to incomplete databases is coNP-complete. The problem can be solved in PTime for a considerable fragment of GN-SQL.

In Section 8, we prove decidability of the *boundedness* problem for GN-Datalog. Boundedness is a classical decision problem in the study of query optimization for recursive queries. It is known to be undecidable for Datalog, but decidable for monadic Datalog. Our result can be viewed as a powerful generalization of the decidability of boundedness for monadic Datalog queries [17].

We conclude in Section 9 by discussing possibilities for further extending GN-SQL and GN-Datalog.

2. PRELIMINARIES

In this section, we review definitions and results concerning the guarded-negation logics GNFO and GNFP from [7]. These results will be put to extensive use in the rest of this paper. We assume familiarity with the basic syntax and semantics of first-order logic.

For clarity, we will maintain a distinction between instances and structures: a structure has an associated domain, which may be a superset of its active domain, and which may depend on the structure in question. Furthermore, structures may interpret not only relation symbols but also constant symbols (which denote, not necessarily distinct, domain elements). Thus, instances may be viewed as a special case of structures, where the domain is the active domain and there are no constant symbols. Unless explicitly stated otherwise (by means of the adjective "unrestricted"), we always assume structures and instances to be finite.

GNFO. Guarded Negation First-Order Logic (GNFO) is the fragment of first-order logic consisting of all formulas built up from atomic formulas (including equalities) using conjunction, disjunction, existential quantification, and guarded negation, that is, negation in the specific form $\alpha \wedge \neg\phi$ where α is an atomic formula (possibly an equality statement) and all free variables of ϕ occur in α. Note that, since the guard α is allowed to be an equality statement, we are essentially able to negate any formula with at most one free variable (by writing $x = x \wedge \neg\phi(x)$). Formally, the formulas of GNFO are generated by the recursive definition

$$\phi ::= R(t_1, \ldots, t_n) \mid t_1 = t_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \exists x\phi \mid \alpha \wedge \neg\phi$$

where each t_i is either a variable or a constant symbol, and, in the last clause, α is an atomic formula containing all free variables of ϕ. Note that function symbols (of arity greater than zero) are not considered.

In the above definition, we required α to be an atomic formula containing all free variables of the negated formula ϕ. Occasionally, it is convenient to allow a slightly more liberal syntax. Let us say that α is a *generalized guard* for ϕ if α is a disjunction of existentially quantified atomic formulas such that the free variables of ϕ are included in the free variables of each disjunct. One could extend GNFO by allowing generalized guards in the definition of guarded negation, thus admitting formulas such as $(\exists uv\, R(x, y, u, v) \vee \exists uv\, R(y, x, u, v)) \wedge \neg Sxy$. This would not increase the expressive power of GNFO: if a negation is guarded by a generalized guard, we can "pull out" the disjunction and the existential quantification to obtain an equivalent formula without generalized guards (at the cost of a possibly exponential blow-up in formula size). In particular, the above example can be equivalently expressed by $\exists uv(R(x, y, u, v) \wedge \neg Sxy) \vee \exists uv(R(y, x, u, v) \wedge \neg Sxy)$. Therefore, for simplicity, our definition of GNFO does not allow for generalized guards.

GNFP. Guarded Negation Fixed Point Logic (GNFP) further extends GNFO with an operator for least fixed points of positively definable monotone operations on relations. That is, we introduce second-order variables (also called fixed-point variables) of arbitrary arity, which may be used to form atomic formulas in the same way as ordinary relation symbols, and if ϕ is any GNFP formula, X an n-ary second-order variable ($n \geq 1$) occurring only positively in ϕ (i.e, under an even number of negations), $\mathbf{x} = x_1, \ldots, x_n$ a sequence of first-order variables, and $\mathbf{t} = t_1, \ldots, t_n$ a sequence of terms (first-order variables or constant symbols), and the free first-order variables of ϕ are included in $\mathbf{x}$, then

$$[\mathrm{LFP}_{X,\mathbf{x}}\ \alpha \wedge \phi](\mathbf{t})$$

is also a formula of GNFP, where α is a generalized guard for ϕ, i.e., a disjunction of existentially quantified atomic formulas (involving only atomic relations, no second-order variables), such that all free first-order variables of ϕ are also free variables of each disjunct of α.

In the above formula, the LFP operator is a generalized quantifier binding the variables X and $\mathbf{x}$. The formula expresses that the tuple $\mathbf{t}$ belongs to the least fixed-point of the monotone operation on relations defined by $\alpha \wedge \phi$. Incidentally, here, unlike in the case of GNFO, it is important that α is allowed to be a *generalized* guard.

In what follows, whenever we consider LFP formulas, we will always assume that they do not have any free second-order variables. The formal semantics of $[\mathrm{LFP}_{X,\mathbf{x}}\ \alpha \wedge \phi](\mathbf{y})$ is the familiar one from least fixed-point logic, cf. [1]. If the formula ϕ has at most one free variable x, we may omit the guard α, which can be assumed to be the equality statement $x = x$. For example, the GNFP formula

$$[\mathrm{LFP}_{X,x}\ P(x) \vee \exists y\ R(x, y) \wedge X(y)](z)$$

says that there is an R-path from z to some element in P.

Definability of Greatest Fixed Points. Besides the above *least fixed-point* operator, we can consider an analogous operator GFP for taking the *greatest fixed point* of a definable monotone operation on relations. However, as it turns out, it is possible to define the GFP operator in terms of the LFP operator (and vice versa) using a dualization via guarded negation. Specifically, $[\mathrm{GFP}_{X,\mathbf{x}}\, \alpha(\mathbf{x}) \wedge \phi(\mathbf{x})](\mathbf{t})$ can be equivalently expressed as $\alpha(\mathbf{t}) \wedge \neg[\mathrm{LFP}_{X,\mathbf{x}} \alpha(\mathbf{x}) \wedge \neg\phi'(\mathbf{x})](\mathbf{t})$, where ϕ' is obtained from ϕ by replacing all subformulas of the form $X(\mathbf{t}')$ by $\alpha(\mathbf{t}') \wedge \neg X(\mathbf{t}')$. For this reason, the above definition of GNFP does not include GFP as a primitive operator.

Definability of Simultaneous Fixed Points. It is common, in the literature on fixed point logics, to consider also a *simultaneous* least fixed point operator, that takes as arguments not a single formula but a tuple of formulas. More precisely, in the context of GNFP it is natural to consider also formulas of the form $[\mathrm{LFP}_{X_i} S](\mathbf{t})$ where

$$S = \begin{cases} X_1(\mathbf{x}_1) & \leftarrow \;\; \alpha_1(\mathbf{x}_1) \wedge \phi_1(X_1, \ldots, X_n, \mathbf{x}_1) \\ & \vdots \\ X_n(\mathbf{x}_n) & \leftarrow \;\; \alpha_n(\mathbf{x}_n) \wedge \phi_n(X_1, \ldots, X_n, \mathbf{x}_n) \end{cases}$$

is a system of GNFP formulas, with each X_k a distinct second-order variable, whose arity matches the length of the tuple $\mathbf{x}_k$, and which occurs only positively in $\phi_1, \ldots, \phi_n$, and where $\mathbf{t}$ is a tuple of terms of the same length as $\mathbf{x}_i$. Here, the system S can be viewed as defining a monotone operation on tuples of relations, and the above formula expresses that $\mathbf{t}$ belongs to the i-th component of the least fixed point of this operation. It is well known that simultaneous fixed point expressions of this form can be expressed equivalently using a nesting of ordinary, single-variable, fixed point operators, possibly at the cost of an exponential blow-up in formula size (cf. for example [2]). Hence, extending GNFP with such a simultaneous least fixed point operator does not increase its expressive power.

Disjunctive Normal Form for GNFO and Width. We say that a GNFO formula is in *Disjunctive Normal Form* (DNF) if it is a disjunction of disjunction-free GNFO formulas, no existential quantifier occurs directly below a conjunction sign, and no conjunction sign occurs directly below a negation sign. Equivalently, a GNFO formula is in DNF if it is a disjunction of GNFO formulas ϕ generated by the following recursive definition:

$$\begin{array}{lcl} \phi & ::= & \exists x_1, \ldots, x_n(\zeta_1 \wedge \cdots \wedge \zeta_m) \\ \zeta & ::= & R(t_1, \ldots, t_n) \;\mid\; (t_1 = t_2) \mid \alpha \wedge \neg\phi \end{array} \qquad (1)$$

where, in the last clause, α is an atomic formula containing all free variables of ϕ. Every GNFO formula is equivalent to one in DNF, of possibly exponential size, that can be obtained by repeatedly applying the following equivalences.

$$\begin{array}{ll} (\exists x\phi) \wedge \psi \simeq \exists x'(\phi[x'/x] \wedge \psi), & \phi \wedge (\psi \vee \chi) \simeq (\phi \wedge \psi) \vee (\phi \wedge \chi) \\ \exists x(\phi \vee \psi) \simeq \exists x\phi \vee \exists x\psi, & \alpha \wedge \neg(\phi \wedge \psi) \simeq (\alpha \wedge \neg\phi) \vee (\alpha \wedge \neg\psi) \end{array}$$

The *width* of a GNFO formula ϕ is the number of variables occurring (free or bound) in the DNF formula obtained from ϕ by applying the above rules.

A *union of conjunctive queries* (UCQ) is a GNFO formula in DNF without negation. Thus, GNFO can be naturally viewed as an extension of UCQs with guarded negation.

Known Decidability and Complexity Results. The following theorem summarizes what is known about GNFO and GNFP that is relevant for present purposes. Recall that the satisfiability problem has as input a formula $\phi(\mathbf{x})$, and asks whether there exists a structure M and a tuple of elements $\mathbf{a}$ such that $M \models \phi(\mathbf{a})$. The entailment problem takes as input two formulas $\phi(\mathbf{x})$, $\psi(\mathbf{x})$, and asks whether it is the case that, for every structure M and for every tuple of elements $\mathbf{a}$, $M \models \phi(\mathbf{a})$ implies $M \models \psi(\mathbf{a})$. The model checking problem has as input a formula $\phi(\mathbf{x})$, a structure M, and a tuple of elements $\mathbf{a}$, and asks whether $M \models \phi(\mathbf{a})$.

Theorem 2.1 ([7])
1. *The satisfiability problem and the entailment problem for GNFO and for GNFP are decidable and 2ExpTime-complete. This holds both for finite structures and for unrestricted structures.*
2. *For GNFO formulas, satisfiability over finite structure coincides with satisfiability over unrestricted structures, and similarly for entailment. The same does* not *hold for GNFP.*
3. *The model checking problem for GNFO is* $\mathrm{P}^{\mathrm{NP}[\log^2]}$*-complete (combined complexity). For GNFP, the problem is* P^{NP}*-hard and is contained in* $\mathrm{NP}^{\mathrm{NP}} \cap \mathrm{coNP}^{\mathrm{NP}}$.

In the above theorem, $\mathrm{P}^{\mathrm{NP}[\log^2]}$ refers to those problems that can be solved by a polynomial time deterministic algorithm that is allowed to ask $O(\log^2(n))$ queries to an NP-oracle, cf. Section 7.1. A close analysis of the 2ExpTime upper bound argument for the satisfiability and entailment problems of GNFP shows that these results extend to the case with simultaneous fixed-point operators (both on finite structures and on unrestricted structures).[1]

2.1 Guarded Negation in Relational Algebra

The concept of guarded negation can be equivalently cast in terms of the Relational Algebra, where negation is expressed by means of the difference operator. Consider the Relational Algebra (RA) defined over a schema consisting of relation symbols of specified arity using the following primitive operators (cf. [1] for their semantics).

Atomic Relations: every relation symbol belongs to RA.
Selection: if $E \in$ RA has arity k and $1 \leq i, j \leq k$, then $\sigma_{i=j}(E)$ belongs to RA and has arity k.
Projection: if $E \in$ RA has arity k and $1 \leq i_1, \ldots, i_n \leq k$, then $\pi_{i_1 \ldots i_n}(E)$ belongs to RA and has arity n.
Crossproduct: if $E_1, E_2 \in$ RA have arity k and n, respectively, then $E_1 \times E_2$ belongs to RA and has arity $k+n$.

[1]Specifically, the proof of the 2ExpTime upper bound for GNFP is based on a satisfiability preserving translation from GNFP to guarded fixed-point logic (GFP). The translation may give rise to an exponential blow-up in the size of the formula, but it preserves the width (following a suitable definition of width, analogous to the definition of width for GNFO formulas). The satisfiability problem for GFP formulas, in turn, is decidable in time $2^{poly(|\phi|) \cdot exp(\mathrm{width}(\phi))}$ (where $poly(n)$ is short for $n^{O(1)}$ and $exp(n)$ is short for $2^{poly(n)}$) by a reduction to the emptiness problem for a suitable type of automata [23, 6]. The translation from GFP formulas to automata extends immediately to the case for formulas containing simultaneous fixed-point operators. Furthermore, the polynomial-time inductive satisfiability-preserving translation from GNFP to GFP given in [7] (which in fact simply commutes with the fixed-point operators) extends in a straightforward manner to the case where the input and output formulas may contain simultaneous fixed-point operators.

$$\begin{array}{lcl} \textit{query} & := & \textsf{select } (t_1 \textsf{ as } \textsc{attr}_1, \ldots, t_n \textsf{ as } \textsc{attr}_n) \textsf{ from } (\textsc{rel}_1\ R_1, \ldots, \textsc{rel}_m\ R_m) \textsf{ where } \textit{condition} \\ & & \mid \ \textit{query} \textsf{ union } \textit{query} \ \mid \ \textit{query} \textsf{ intersect } \textit{query} \ \mid \ \textit{query} \textsf{ except } \textit{query} \\ \textit{condition} & := & \textsf{true} \ \mid \ t_1 = t_2 \ \mid \ t \textsf{ in } \textit{query} \ \mid \ \textsf{exists}(\textit{query}) \\ & & \mid \ \textit{condition} \textsf{ and } \textit{condition} \ \mid \ \textit{condition} \textsf{ or } \textit{condition} \ \mid \ \textsf{not}(\textit{condition}) \end{array}$$

Figure 1: Grammar for FO-SQL queries

Union, Intersection, and Difference: if $E_1, E_2 \in$ RA both have arity k, then $E_1 \cup E_2$, $E_1 \cap E_2$ and $E_1 - E_2$ belong to RA and have arity k.

Codd's completeness theorem states that RA has the same expressive power as the domain-independent fragment of first-order logic, cf. [1]. Let us briefly recall here the definition of domain independence for first-order formulas without constant symbols [1]. The *active domain* of a structure M is the set $adom(M)$ of all elements that occur in a tuple belonging to one of the relations. For any structure M, let M' be a copy of the same structure but where all elements outside $adom(M)$ are removed. A first-order formula $\phi(\mathbf{x})$ without constant symbols is *domain-independent* if (i) whenever $M \models \phi(\mathbf{a})$, then the tuple $\mathbf{a}$ consists of elements of $adom(M)$, and (ii) for all tuples $\mathbf{a}$ consisting of elements of $adom(M)$, $M \models \phi(\mathbf{a})$ if and only if $M' \models \phi(\mathbf{a})$. The same definition applies to formulas with fixed-point operators. Examples of first-order formulas that are *not* domain-independent are $P(x) \vee Q(y)$, $x = x$, and $\neg P(x)$.

We say that a relation algebra expression is *negation-guarded* if every occurrence of the difference operator is of the form $\pi_{i_1 \ldots i_m}(R) - E$ where R is a relation symbol. We denote by GN-RA the negation-guarded fragment of RA. It can be shown by straightforward inductive translations that GN-RA captures GNFO in the following sense.

Theorem 2.2 *Every k-ary GN-RA expression is equivalent to a domain-independent GNFO formula $\phi(x_1, \ldots, x_k)$, and vice versa, via a linear translation from GN-RA to GNFO and an exponential translation backwards.*

Let R, S be relation symbols of arity 2 and 1, respectively. The following RA expressions are *not* negation-guarded.

$(\pi_1(R) \times S) - \pi_{1,1}(R)$ (distinct pairs from $\pi_1(R) \times S$)

$\pi_{1,4}(\sigma_{2=3}(R \times R)) - R$ (reachability in two steps, not one)

$\pi_1(R) - \pi_1((\pi_1(R) \times S) - R)$ (the quotient $R \div S$)

In fact, it follows from results in [7] that none of these expressions is equivalent to a GN-RA expression.

Observe that in the above definition of GN-RA we did not allow for the use of constant values in selections and projections. This was only to simplify presentation. All complexity results that we will present go through in the presence of constant values, cf. Section 9.

3. GUARDED NEGATION IN SQL

In this section, we discuss what it means for an SQL query to have guarded negation. More precisely, we consider a simple, first-order expressively complete, fragment of SQL with a set-based semantics, that we call FO-SQL, and we characterize the queries in this fragment that can be expressed in GNFO.

FO-SQL: a Simple First-Order Fragment of SQL. In this section, unlike in the rest of the paper, we work with named schemas. A *named schema* is a collection of relation names, each with an associated list of attribute names. For the discussion below, assume we have a fixed schema, say, consisting of BOOK(ISBN,AUTHOR,TITLE) and LOCATION(ISBN,SHELF,NUMBER). We also fix an infinite supply of "tuple variables" (also known as *aliases*, and denoted by $R_1, R_2, \ldots$). By a *term* t we will mean an expression of the form R_i.ATTR where R_i is a tuple variable and ATTR is an attribute name.

We consider SQL expressions that are generated by the simple grammar given in Figure 1, where each t_i is a term, each $\textsc{rel}_i$ is a relation name, $\textsc{attr}_1, \ldots, \textsc{attr}_n$ are distinct attribute names, and $R_1, \ldots, R_m$ are distinct tuple variables. This grammar generates queries that may have free tuple variables, i.e., there may be occurrences of tuple-variables R_i that are not in the scope of a select-from-where clause where they are declared. We will refer to queries with free tuple variables as *open queries* (or *correlated subqueries*), and we refer to queries without free tuple variables as *closed queries* (or *uncorrelated subqueries*). We will denote by $FV(q)$ the set of free tuple variables of q. We will be mainly interested in closed queries. Note, however, that closed queries *are* allowed to contain subexpressions of the form exists(q) or of the form t in q where q is an open query.

We only consider queries that are well-typed in the sense that each (open or closed) query can be consistently assigned a (unique) *type*, which is a list of attribute names, where

1. the type of a select-from-where query is the set of attribute names specified in its select clause;
2. the union, intersect, and except operators take as arguments two queries of equal type, yielding a query of the same type.

Furthermore, terms R_i.ATTR are only allowed to occur when ATTR belongs to the schema of the relation to which the occurrence of R_i in question is bound, and conditions of the form t in q are allowed only when q is a unary query, i.e., when the type of q consists of a single attribute.

By an FO-SQL query, we will mean a closed query satisfying the above requirements. Two examples are given in Figure 2. We assume that the reader is familiar with the semantics of SQL, and hence omit the formal semantics of the fragment FO-SQL. We just mention that we disregard order and duplicates, treating relations as *sets* of tuples. It is known that, under this set-based semantics, FO-SQL is expressively complete for first-order logic, in the sense of Codd's expressive completeness theorem [1, 29]. That is, FO-SQL queries have the same expressive power as the domain-independent fragment of first-order logic. Since FO-SQL queries are defined in terms of named schemas, while the syntax of first-order logic is based on unnamed schemas in which the attributes of a relation are identified by natural numbers instead of by attribute names, here, we consider a

select A.NAME from AUTHOR A where not exists(
  select B.TITLE from BOOK B where B.AUTH = A.NAME)

select A.NAME from AUTHOR A where not exists(
  select B.TITLE from BOOK B where not B.AUTH = A.NAME)

Figure 2: Two examples of FO-SQL queries, the first negation-guarded and the second not.

FO-SQL query q of type $\{A_1, \ldots, A_n\}$ to be "equivalent" to a first-order formula $\phi(x_1, \ldots, x_n)$, containing the relation names REL_i occurring in q as relation symbols of appropriate arity, if for every instance I and for every n-tuple $\mathbf{a}$, the tuple $\mathbf{a}$ is an answer to q in I if and only if $I \models \phi(\mathbf{a})$.

The most important features of (full) SQL that are excluded in the above definition of FO-SQL are *constants*, *arithmetical comparison*, and *aggregation*. We will discuss the importance of these restrictions later in Section 9.

GN-SQL: the Guarded-Negation Fragment of FO-SQL. We say that an SQL query is *negation-guarded* if the following two conditions hold:

1. each except operator has as its first argument a simple projection and as its second argument an uncorrelated subquery.
2. each not operator has as its argument a condition with at most one free tuple variable.

Here, by a *simple projection*, we mean a select-from-where query, where the where-clause is 'true'. GN-SQL is the fragment of FO-SQL consisting of all negation-guarded queries.

To illustrate this definition, consider the two queries given in Figure 2. The first query involves a single occurrence of negation, which is guarded, since the negated condition has only one free tuple variable, namely A. The second query, on the other hand, is *not* a GN-SQL query, since the second occurrence of negation is not guarded. Indeed, the condition B.AUTH = A.NAME has two free tuple variables.

The next theorem states that GN-SQL captures GNFO, in the same way that FO-SQL captures full first-order logic, as we discussed above (the same conventions apply, concerning what it means for a FO-SQL query to be equivalent to a first-order formula).

Theorem 3.1 (GN-SQL is Codd-complete for GNFO) *Each GN-SQL query can be translated in linear time into an equivalent domain-independent GNFO formula. Conversely, each domain-independent GNFO formula can be translated in exponential time into an equivalent GN-SQL query.*

It can be shown that the exponential complexity of the translation from GNFO to GN-SQL is in general unavoidable for formulas of the form $(R(x_1) \vee S(x_1)) \wedge \cdots \wedge (R(x_n) \vee S(x_n))$. On the other hand, the proof of Theorem 3.1 shows that if the schema includes a unary relation ADOM that is guaranteed to denote the active domain of the instance, then there is a polynomial translation.

3.1 Negation in Practice: a Benchmark Study

In order to assess the usage of negation in SQL queries in practice, we have studied the workloads of two standard SQL benchmarks, namely TPC-H [37] and TPC-DS [36]. These benchmarks were designed to evaluate and compare the performance of relational database management systems. In addition, we studied the sample queries published on the Sloan Digital Sky Survey (SDSS) SkyServer website [35], a selection of actual queries submitted by SDSS users. For each query, we investigated whether the query uses negation, and, if so, whether the query is negation-guarded. We also studied the use of inequalities, and investigated which of these inequalities can be expressed using guarded negation. The results, given in Figure 3, shows that most queries using negation use only guarded negation. We should note here that most queries contain SQL constructs, such as aggregation, that do not belong to FO-SQL. Therefore, the queries are not necessarily expressible in GN-SQL. The statistics in Figure 3 are only concerned specifically with the explicit use of *negation*. We also did not investigated the use of other SQL constructs such as *outer joins*, that can, in some sense, be viewed as involving an implicit form of negation.

benchmark	queries	queries with negation[1]	queries with unguarded negation	queries with inequalities[2]	queries with unguarded inequalities
TPC-H	22	4	0	3	1
TPC-DS	99	8	1	8	7
SkyServer	48	2	0	8	1

[1] By negation, we mean any occurrence of not or except.
[2] An inequality is any occurrence of <> or !=. An inequality is *guarded* if the corresponding negation not(... = ...) is guarded.

Figure 3: Usage of negation in SQL benchmarks

4. GUARDED NEGATION IN DATALOG

In this section, we present a powerful variant of Datalog with stratified negation, which we call GN-Datalog and which, in terms of its expressive power, is contained in GNFP. We first briefly recall the syntax and semantics of Datalog, with and without stratified negation.

Definition 4.1 (Datalog) A *Datalog program* is specified by a triple $\Pi = (\text{EDB}^\Pi, \text{IDB}^\Pi, \text{Rules}^\Pi)$, where EDB^Π and IDB^Π are disjoint sets of relation names, each with an associated arity, and Rules^Π is a finite set of rules of the form

$$\phi \leftarrow \psi_1, \ldots, \psi_n$$

where $\phi, \psi_1, \ldots, \psi_n$ are atomic formulas of the form $R(x_1, \ldots, x_n)$ with $R \in \text{EDB}^\Pi \cup \text{IDB}^\Pi$ and $x_1, \ldots, x_n$ a sequence of first-order variables of appropriate length. We refer to ϕ as the head of the rule, and $\psi_1, \ldots, \psi_n$ as the body of the rule. In addition, we require that (i) every first-order variable occurring in the head of a rule must occur in the body, and (ii) the relation in the head of each rule must be an IDB relation.

A Datalog query is a pair (Π, Ans), where Π is a Datalog program and Ans is a union of conjunctive queries over the schema $\text{EDB}^\Pi \cup \text{IDB}^\Pi$. The semantics of a Datalog query is defined as follows: first, if Π is a Datalog program, I an instance over the schema EDB^Π, and k a natural number, then we denote by $\Pi^k(I)$ the instance over the schema $\text{EDB}^\Pi \cup \text{IDB}^\Pi$ containing all facts that can be derived from the facts in I using at most k rounds of applications of rules of Π. In addition, we denote by $\Pi^\infty(I)$ the union $\bigcup_k \Pi^k(I)$. If $q = (\Pi, Ans)$ is a Datalog query and I an instance over the schema EDB^Π, then we denote by $q(I)$ the set of all tuples that are an answer to the query Ans in $\Pi^\infty(I)$.

We remark that the above definition differs slightly from the standard presentation of Datalog. Usually, Ans is required to be a designated relation from IDB^Π instead of a union of conjunctive queries. The presentation we use here is convenient as it helps simplify the definitions below. On the other hand, note that this is not essential: a Datalog program can always be extended with an additional IDB relation and with additional rules computing the Ans query.

Definition 4.2 (Datalog with Stratified Negation) A *Datalog*$^\neg$ *program* is a Datalog program Π where the body of each rule may, in addition, contain atomic formulas of the form $\neg R(x_1, \ldots, x_n)$ provided that $R \in \mathrm{EDB}^\Pi$, and provided that each first-order variable occurring in the head or body of the rule occurs positively in the body. A *Datalog program with stratified negation* is a sequence $\tilde{\Pi} = (\Pi_1, \ldots, \Pi_n)$ of Datalog$^\neg$ programs, called strata, with $n \geq 1$, where for each $i = 2 \ldots n$, $\mathrm{EDB}^{\Pi_i} = \mathrm{EDB}^{\Pi_{i-1}} \cup \mathrm{IDB}^{\Pi_{i-1}}$. We use $\mathrm{EDB}^{\tilde{\Pi}}$ and $\mathrm{IDB}^{\tilde{\Pi}}$ to denote EDB^{Π_1} and $\bigcup_{i=1\ldots n} \mathrm{IDB}^{\Pi_i}$, respectively.

A *Datalog query with stratified negation* is a pair $(\tilde{\Pi}, Ans)$, where $\tilde{\Pi} = (\Pi_1, \ldots, \Pi_n)$ is a Datalog program with stratified negation and Ans is a union of conjunctive queries over the schema $\mathrm{EDB}^{\tilde{\Pi}} \cup \mathrm{IDB}^{\tilde{\Pi}}$. The semantics of Datalog programs and of Datalog queries extends naturally to Datalog with stratified negation, by defining $\tilde{\Pi}^\infty(I)$ as $\Pi_n^\infty(\Pi_{n-1}^\infty(\cdots \Pi_1^\infty(I) \cdots))$ for $\tilde{\Pi} = (\Pi_1, \ldots, \Pi_n)$.

We say that a Datalog program Π is *non-recursive* if no IDB occurs in the body of any of its rules, and hence, in particular, for all instances I we have that $\Pi^\infty(I) = \Pi^1(I)$. We say that a Datalog program with stratified negation is non-recursive if it consists entirely of non-recursive strata.

Definition 4.3 (GN-Datalog) A GN-Datalog program is a Datalog program with stratified negation $\tilde{\Pi} = (\Pi_1, \ldots, \Pi_n)$, where each rule

$$\phi_0 \leftarrow (\neg)\phi_1, \ldots, (\neg)\phi_n \quad \in \mathrm{Rules}^{\Pi_k} \quad (1 \leq k \leq n)$$

is *negation guarded*, meaning that the following holds:

> For each atom ϕ_i that either occurs negated in the body or is the head, the body includes a positive atom ϕ_j containing all first-order variables occurring in ϕ_i, and ϕ_j uses a relation from EDB^{Π_k}.[2]

A *GN-Datalog query* is a Datalog query with stratified negation, where each rule is negation guarded. Note that this requirement concerns only the rules; the answer query Ans can be any union of conjunctive queries.

Theorem 4.4 (Non-recursive GN-Datalog is Codd-complete for GNFO) *Each non-recursive GN-Datalog query is equivalent to a domain-independent GNFO formula, and vice versa, via exponential translations.*

[2]To understand why this is the appropriate definition of negation guardedness, observe that a rule of the form $\phi \leftarrow \psi_1, \ldots, \psi_n$ expresses that $\neg\exists \mathbf{x}(\psi_1 \wedge \cdots \wedge \psi_n \wedge \neg\phi)$, i.e., the head of the rule plays the same role as a negated atom in the body.

The translation from non-recursive GN-Datalog to GNFO given in the proof of Theorem 4.4 can be extended in a straightforward manner to a translation from GN-Datalog to the extension of GNFP with simultaneous fixed-point operators. Since simultaneous fixed-point operators can be eliminated (at the cost of an additional exponential blow-up, cf. Section 2), we obtain following:

Theorem 4.5 *Each GN-Datalog query is equivalent to a domain-independent alternation-free GNFP formula.*

The translation from non-recursive GN-Datalog to GNFO provided by Theorem 4.4 involves an exponential blow-up, due to an elimination of subformula sharing. The translation from GN-Datalog to GNFP provided by Theorem 4.5 involves another exponential blow-up, due to the elimination of simultaneous fixed-point operators. These sources of exponential complexity can be avoided (i) if we transcribe GN-Datalog queries into GNFP formulas over a larger schema (containing a relation symbol not only for each EDB of the GN-Datalog query, but also for each IDB), and (ii) freely use simultaneous fixed-point operators in the GNFP formula. More precisely, the proof of Theorem 4.5 can be adapted in a straightforward manner to show the following result, which will be useful later on (where, for two schemas $S \subseteq \widehat{S}$, an $\widehat{S}$*-expansion* of an instance I over $\widehat{S}$ is an instance over $\widehat{S}$ that agrees with I on all facts over S).

Theorem 4.6 *For every k-ary GN-Datalog query q over a schema S one can compute in polynomial time a GNFP sentence ϕ_q and a GNFP formula $\psi_q(x_1, \ldots, x_k)$, both with simultaneous fixed point operators, and over a possibly larger schema $\widehat{S}$, such that*

1. *each instance I has a unique $\widehat{S}$-expansion $\widehat{I}$ satisfying ϕ_q.*
2. *for all instances I and k-tuples $\boldsymbol{a}$, $\boldsymbol{a} \in q(I)$ iff $\widehat{I} \models \psi_q(\boldsymbol{a})$.*

5. RELATIONSHIPS WITH EXISTING LANGUAGES

Monadic Datalog is a well-known Datalog fragment that combines an interesting level of expressiveness with good algorithmic behavior thanks to a tight connection with tree automata, which also make monadic Datalog suitable for a number of applications, e.g. [21]. It also stands out a fragment for which the boundedness problem is decidable [17], Theorem 8.2 below. Monadic Datalog does not allow any form of negation and since all IDB predicates are unary, guardedness of rule heads is guaranteed, so that monadic datalog rules are trivially negation guarded. We will show that boundedness remains decidable for GN-Datalog.

Datalog LIT is a fragment of stratified Datalog whose model checking has linear-time data complexity [20]. Each Datalog LIT rule must either contain in its body as 'guard' a positive literal containing all variables occurring in the rule, or must solely be comprised of unary literals (including its head). While the 'guard' of guarded rules need not be an EDB atom, [20] shows that every Datalog LIT program can (in exponential time) be transformed into an equivalent one having only EDB atoms as guards. The latter are trivially negation guarded. Every Datalog LIT program is thus equivalent to a GN-Datalog program.

GNFO subsumes a number of formalisms having currency in ontological reasoning, such as the linear- and guarded tuple generating dependencies (tgds) underlying the recently promoted Datalog$^{\pm}$ [12] framework and the more general frontier-guarded tgds [5] that subsume the description logics DL-Lite$_R$ (which captures RDF Schema), $\mathcal{ELI}$, and $\mathcal{ELH}^{dr}_{\bot}$ [3], which is the core of the proposed OWL-EL profile of the OWL 2.0 ontology language. GNFO can encode query answering and containment assertions involving such specifications of constraints or TBoxes. A tgd is a sentence

$$\forall \mathbf{x}, \mathbf{y}\ \phi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ \psi(\mathbf{y}, \mathbf{z}) \qquad (2)$$

where both ϕ and ψ, called the body and the head, respectively, of the tgd rule, are conjunctions of positive atoms. When working under OWA one can assume, as a matter of convenience, that the head of every tgd is a single atom. A tgd is linear if ϕ consists of a single atom; it is guarded if ϕ contains as conjunct an atom $R(\mathbf{x}, \mathbf{y})$, the 'guard', in which all of the variables of the body occur together; and it is frontier-guarded if the body contains an atom $P(\mathbf{y})$ in which all of the variables shared by the body and the head of the rule occur together. Every frontier-guarded tgd naturally translates to a GNFO sentence.

Other query languages that can be viewed as fragments of GNFO include the semi-join algebra [28], as well as Unary Conjunctive View Logic (UCV) and Core XPath, cf. [14].

6. QUERY CONTAINMENT

We now exploit the connection with GNFO and GNFP to show that query containment is decidable for GN-SQL and for GN-Datalog. Recall that a query q is satisfiable if there exists an instance I such that the set of answers $q(I)$ is non-empty, and that a query q_1 is contained in a query q_2 if, for all instances I, $q_1(I) \subseteq q_2(I)$. The satisfiability problem can be viewed as (the complement of) a special case of the query containment problem, where the second query q_2 is any fixed unsatisfiable query.

Theorem 6.1 *Query containment is 2ExpTime-complete for both GN-SQL queries and GN-Datalog queries. Hardness holds already for satisfiability of non-recursive GN-Datalog, and GN-SQL, over a fixed EDB schema.*

The 2ExpTime upper bounds for GN-SQL follow directly from Theorem 3.1 and Theorem 2.1. The 2ExpTime upper bounds for GN-Datalog do not follow directly from Theorem 4.5 and Theorem 2.1, due to the exponential complexity of the translation from GN-Datalog to GNFP involved. However, it follows using Theorem 4.6: let q_1, q_2 be k-ary GN-Datalog queries ($k \geq 0$), and let $\phi_1, \psi_1(x_1, \ldots, x_k)$ and $\phi_2, \psi_2(x_1, \ldots, x_k)$ be the GNFP-formulas with simultaneous fixed point operators obtained by Theorem 4.6. We may assume without loss of generality that the only relation symbols that ϕ_1, ψ_1 and ϕ_2, ψ_2 have in common are the relation symbols that appear in q_1 and q_2. It follows that q_1 is contained in q_2 if and only if $\phi_1 \wedge \psi_1(x_1, \ldots, x_k) \models \phi_2 \rightarrow \psi_2(x_1, \ldots, x_k)$. This gives us the desired result, since, as we explained in Section 2, the 2ExpTime upper bound for GNFP entailment from Theorem 2.1 extends to the case with simultaneous least-fixed point operators. The lower bounds are obtained by adapting the proof of an 2ExpTime-hardness result for a fragment of GNFO in [14].

Theorem 6.1 generalizes the known decidability result for monadic datalog and unions of conjunctive queries [17]. In addition, it easily implies the decidability of containment of Datalog queries in Unions of Conjunctive Queries [15]. This can be seen as follows. For each Datalog query q, let $\widehat{q}$ be the GN-Datalog query obtained from q by guarding each rule using an additional conjunct that is a fresh EDB relation. Then for each UCQ q' over the original schema, we have that q is contained in q' if and only if $\widehat{q}$ is contained in q'. One direction follows directly from the fact that $\widehat{q}$ is contained in q. For the other direction, note that every counterexample I to the containment of q in q' gives rise to a counterexample I' to the containment of $\widehat{q}$ in q'. The instance I' in question extends I by interpreting each new EDB relation as the total relation containing all tuples over the active domain of I.

As a direct consequence of the finite model property of GNFO [7] and of Theorems 3.1 and 4.4, respectively, we find that query containment is *finitely controllable* for GN-SQL queries and for non-recursive GN-Datalog queries. By this we mean that one query is contained in an other on finite instances if, and only if, the containment holds on unrestricted instances (a finite model property).

Theorem 6.2 *Satisfiability and containment are finitely controllable for GN-SQL and for non-recursive GN-Datalog.*

7. QUERY ANSWERING

7.1 (Closed-World) Query Evaluation

Since GN-SQL and non-recursive GN-Datalog admit translations into first-order logic, the *data complexity* of query evaluation is in AC^0 for both query languages. Similarly, since GN-Datalog is contained in the fixed-point logic FO(LFP), the data complexity of query evaluation is in PTime. In fact, there is a GN-Datalog query (a monadic Datalog query) for which query evaluation is PTime-hard in terms of data complexity [20].

In what follows we consider the *combined complexity* of query evaluation. Datalog evaluation is known to be ExpTime-complete for combined complexity (implicit in [39]). Monadic Datalog evaluation is known to be NP-complete [21]. The "guarded fragment of Datalog" (every rule contains an EDB atom containing all variables occurring in the rule) evaluation is in PTime [20]. Non-recursive Datalog with stratified negation is PSPACE-complete [18].

Recall that $\mathrm{P}^{\mathrm{NP}[\log^2]}$ is the class of those problems that can be solved by a polynomial time deterministic algorithm that is allowed to ask $O(\log^2(n))$ queries to an NP-oracle. (It relates to better known complexity classes this way: NP $\subseteq$ DP $\subseteq \mathrm{P}^{\mathrm{NP}\ [\log]} \subseteq \mathrm{P}^{\mathrm{NP}[\log^2]} \subseteq \cdots \subseteq \mathrm{P}^{\mathrm{NP}\ [\log^i]} \subseteq \mathrm{P}^{\mathrm{NP}} \subseteq \Sigma^p_2 \subseteq$ PSPACE $\subseteq$ EXPTIME.)

Theorem 7.1 *The combined complexity of evaluating GN-SQL queries is* $\mathrm{P}^{\mathrm{NP}[\log^2]}$*-complete.*

Proof. The upper bound follows directly from Theorem 3.1 and Theorem 2.1. For the lower bound, observe that the translation from GNFO to GN-SQL given in the proof of Theorem 3.1 is polynomial in the presence of a unary relation ADOM containing all elements in the active domain. We may assume without loss of generality that our input instance contains such a relation. Therefore, the lower bound from Theorem 2.1 extends to GN-SQL as well. □

We show here that the same problem is P^{NP}-complete for GN-Datalog. Recall that the best known upper bound on the complexity of model checking GNFP is $\mathrm{NP}^{\mathrm{NP}} \cap \mathrm{coNP}^{\mathrm{NP}}$.

Theorem 7.2 *The combined complexity of evaluating GN-Datalog queries is* P^{NP}*-complete. Hardness holds already for non-recursive GN-Datalog queries with only unary IDB predicates and nullary negation.*

7.2 Open-World Query Answering

Open world (OWA) query answering is the following problem: *given a query q, an instance I, and a tuple of values $\mathbf{a}$, decide whether it is the case that $\mathbf{a}$ belongs to the answers of q in every instance extending I with additional facts.* An instance of open-world query answering $I \models_{\mathrm{OWA}} q(\mathbf{a})$ thus asks for the unsatisfiability of $I \cup \{\neg q(\mathbf{a})\}$ in the usual first-order semantics, when treating I as a set of atomic facts with its elements as constants. Open world semantics is the natural choice when working with incomplete databases, in data exchange settings, and in the context of ontological reasoning. In each of these settings, open world query answering is an extensively researched problem.

In this section, we investigate the data complexity of open-world query answering for queries with guarded negation. Formally, for each query q we denote by OWA_q the problem, given an instance I and a tuple of values $\mathbf{a}$ from $adom(I)$, to decide whether $I \models_{\mathrm{OWA}} q(\mathbf{a})$. More generally, for each query q and for each set of constraints Σ, we denote by $\mathrm{OWA}_{q,\Sigma}$ the problem, given an instance I, to decide whether $I, \Sigma \models_{\mathrm{OWA}} q$.

Note that, in the absence of constraints, for conjunctive queries q, by monotonicity, the problem $I \models_{\mathrm{OWA}} q(\mathbf{a})$ coincides with $I \models q(\mathbf{a})$, and therefore OWA_q is in PTime (in fact, in AC^0). For first-order queries q, on the other hand, the problem OWA_q can be undecidable. We will show below that the problem is decidable for first-order queries with guarded negation.

As constraints, we will consider tuple-generating dependencies, cf. (2), and key constraints. As noted above, linear-, guarded- and frontier-guarded tgds [5, 12] are expressible in GNFO. With respect to OWA query answering, conjunctive queries are known to be FO-rewritable relative to linear tgds [12] and possess Datalog rewritings relative to frontier-guarded tgds [4]. Accordingly, the data complexity of open-world query answering for conjunctive queries against linear tgds is in AC_0, in PTime for frontier-guarded tgds, and PTime-complete already for guarded tgds [12].

We begin by observing that OWA query answering for GNFO queries, as for many description logics [34, 13, 32], has coNP data complexity. For an instance I, we denote by $|I|$ the total number of facts of I, and, for two instances I, J, we write $I \subseteq J$ if every fact of I is a fact of J.

Proposition 7.3 *Let $\phi(\boldsymbol{x})$ be a fixed GNFO formula. For an instance I and a tuple $\boldsymbol{a}$ of elements from $adom(I)$, if there is an instance $J \models \phi(\boldsymbol{a})$ with $I \subseteq J$, then there is an instance $J \models \phi(\boldsymbol{a})$ with $I \subseteq J$ and $|J| = O(|I|)$.*

Proposition 7.3 tells us that, in solving the open-world query answering problem for GNFO queries, it suffices to consider ony instances whose size is linear in the size of the input instance. This gives us the following:

Theorem 7.4 *For each GNFO query q (in particular, for each GN-SQL query), OWA_q is in coNP. There is a boolean GN-SQL query q for which OWA_q is coNP-hard.*

Proof. The coNP upper bound is immediate from the above proposition. Given an instance I with distinguished elements $\mathbf{a}$, Proposition 7.3 shows that to refute $I \models_{\mathrm{OWA}} q(\mathbf{a})$ it suffices to guess a linear size instance J with $I \subseteq J$ and test in polynomial time that J satisfies $\neg q(\mathbf{a})$. The lower bound is established by a reduction from 3-colorability. Let q be the GNFO sentence (for readability, we omit the repeated occurrences of Nx as guard):

$$\exists x(Nx \wedge \neg P_1 x \wedge \neg P_2 x \wedge \neg P_3 x) \;\vee\; \bigvee_i \exists xy(Exy \wedge P_i x \wedge P_i y) \qquad (3)$$

expressing that P_1, P_2, P_3 do not constitute a valid 3-coloring of the graph (N, E). It is easy to check that a simple undirected graph G is not 3-colorable iff $G \models_{\mathrm{OWA}} q$, and it is straightforward to formulate the domain-independent boolean query (3) in GN-SQL. □

This is remarkable, given that open-world query answering is in general undecidable for first-order queries, even in the absence of constraints.

Recall that every frontier-guarded tgd can be formulated as a GNFO sentence. This allows us to lift the above result to the open-world query answering problem with constraints that are frontier-guarded tgds. More precisely, if Σ is a set of frontier-guarded tgds, then $\mathrm{OWA}_{q,\Sigma}$, by definition, coincides with $\mathrm{OWA}_{q \vee \bigvee_{\sigma \in \Sigma} \neg\sigma}$, and therefore we get the following.

Corollary 7.5 *For each GNFO query q and for each finite set of frontier-guarded tgds Σ, $OWA_{q,\Sigma}$ is in coNP.*

In various contexts, such as data exchange [19], it is useful to consider incomplete databases that contain, besides constant values, also labeled null values. In this case, open world query answering is defined not in terms of extensions of instances, but in terms of homomorphisms that are allowed to map the labeled null values to constant values or to other labeled null values. It is worth observing that the above proofs go through in this more general setting with null values, showing that for GNFO queries q and for finite sets of frontier-guarded tgds Σ, $\mathrm{OWA}_{q,\Sigma}$ is in coNP even over instances containing labeled nulls.

Next we identify a subfragment of GNFO that accommodates the earlier mentioned formalisms including conjunctive queries and frontier-guarded tgds and whose queries enjoy PTime data complexity for OWA. Recall that open-world query answering $I \models_{\mathrm{OWA}} q$ asks for the unsatisfiability of $I \cup \{\neg q\}$. Under negation, the subformula $\exists x(Nx \wedge \neg P_1 x \wedge \neg P_2 x \wedge \neg P_3 x)$ of the coNP-complete query (3) turns into the disjunctive requirement $\forall x(Nx \rightarrow P_1 x \vee P_2 x \vee P_3 x)$ that is, in an intuitive sense, ultimately responsible for intractability. Indeed, it has been observed in the context of DL-Lite that the introduction of even the weakest form of disjunction renders query answering intractable (see, e.g., [13]). It turns out that the positive occurrence of conjunctions $\neg A(x) \wedge \ldots \wedge \neg B(x)$ involving two or more negated conjuncts are the only source of intractability in GNFO queries.

Definition 7.6 (serial GNFO queries, SGNQ) A GNFO-formula φ is *serial* if it is in DNF and no conjunction $\neg\chi(x) \wedge \ldots \wedge \neg\psi(x)$ with two or more negated conjuncts occurs positively in φ, i.e., in the scope of an even number of negations. Let SGNQ denote the set of serial GNFO queries.

Clearly, every union of conjunctive queries is a serial GNFO query. Furthermore, every frontier-guarded tgds, as well as its negation, is equivalent to a boolean serial GNFO queries. It fact, for every finite set Σ of frontier-guarded tgds and for every serial GNFO query q, we have that $q \vee \bigvee_{\sigma\in\Sigma} \neg\sigma$ is a serial GNFO query. In other words, the reduction from open-world query answering in the presence of frontier-guarded tgds to open-world query answering in the absence of tgds, that we gave earlier, holds also in the case of serial GNFO queries.

Theorem 7.7 *For each SGNQ q and for each finite set Σ of frontier-guarded tgds, $OWA_{q,\Sigma}$ is in PTime.*

In fact, for every boolean SGNQ q we can effectively compute a boolean Datalog query q' such that for all instances I, we have $I \models_{\text{OWA}} q \iff I \models q'$.

There is a boolean SGNQ query q for which OWA_q is PTime-complete.

The proof is based on a reduction from the open-world query answering problem for SGNQs in the presence of frontier-guarded tgds to the open-world query answering problem for conjunctive queries in the presence of frontier-guarded tgds. A PTime solution of the latter problem via Datalog rewritings is due to [4].

Finally, we show that OWA answering GNFO queries under key constraints is undecidable. This holds even for a fixed GNFO query and a fixed key constraint of the form $\forall\mathbf{x}yz(F(\mathbf{x},y) \wedge F(\mathbf{x},z) \rightarrow y = z)$ with F a relation symbol.

Theorem 7.8 *(i) There is a boolean conjunctive query q and a set Σ comprising guarded tgds and a single key constraint, so that $OWA_{q,\Sigma}$ is undecidable.*

(ii) There is a boolean SGNQ q and a key constraint σ, so that $OWA_{q,\{\sigma\}}$ is undecidable.

While undecidability of the uniform problem (where the query is part of the input) follows from various similar results for weaker formalisms [34], for a fixed query this seems to be a new result.

8. BOUNDEDNESS AND FIRST-ORDER DEFINABILITY

In this section, we study the boundedness problem for GN-Datalog. Our main result, Corollary 8.9, states that it is decidable whether a GN-Datalog program is fully bounded, i.e., whether, for every instance, the computation of each stratum of the GN-Datalog program reaches a fixed point in a bounded number of steps.

The semantics of a Datalog program Π can be defined in terms of a *least fixed point* for the IDB predicates. For this we view Π, or rather each of its instatiations Π_I over a given instance I, as a monotone operator. An application of this operator to any instantiation of the IDB predicates produces the result of firing all rules once and in parallel, on these IDB predicates and the static EDB predicates as given in I. This operator Π_I is monotone, and the desired interpretation of the IDB predicates in $\Pi^\infty(I)$ is its unique least fixed point. This view extends to not necessarily finite instances I, where Π_I, due to its monotonicity, still has a unique least fixed point, also refered to as $\Pi^\infty(I)$. As in the case of finite instances, this fixed point is obtained as the limit of the monotone sequence of stages Π_I^α generated by iterating Π_I as an update operator, starting from the empty instantiation for all IDB predicates in stage 0, and taking unions at limit ordinals, until finally (for cardinality reasons) a stage Π_I^α is reached that is a fixed point, and indeed the unique least fixed point ($\Pi_I^{\alpha+1} = \Pi_I^\alpha$ implies $\Pi_I^\alpha = \Pi^\infty(I)$).

All these considerations hold for any notion of program or recursion scheme that shares the crucial monotonicity with Datalog programs. Monotonicity refers to monotonicity in the IDB arguments, and is guaranteed by syntactic positivity in the IDB predicates in all cases we consider.We are mostly interested in IDB-positive GNFO-programs, which we first investigate in isolation, towards understanding their stratified, and overall no longer monotone, use in GN-Datalog (cf. Definition 8.4 below).

The notion of *boundedness* captures the semantic and procedural essence of non-recursive behavior (in contrast with syntactic non-recursiveness as defined in Section 4, which focuses on a trivial reason for boundedness).

Definition 8.1 A monotone program Π is *c-bounded* (bounded in the classical sense, or over unrestricted instances) if there exists some $n \in \mathbb{N}$ such that $\Pi_I^{n+1} = \Pi_I^n$ for every finite or infinite instance I. It is bounded over a class of instances $\mathcal{I}$ if there is such an n that is good for all $I \in \mathcal{I}$. We call Π *f-bounded* if it is bounded over the class of *all finite* instances.
$\text{BDD}(\mathcal{P},\mathcal{I})$ stands for the *boundedness problem* for programs from $\mathcal{P}$ over instances from $\mathcal{I}$: given $\Pi \in \mathcal{P}$, decide whether Π is bounded over $\mathcal{I}$. We reserve the names $\text{BDD}_c(\mathcal{P})$ and $\text{BDD}_f(\mathcal{P})$ for $\text{BDD}(\mathcal{P},\text{ALL})$ and $\text{BDD}(\mathcal{P},\text{FIN})$, where ALL and FIN are the classes of unrestricted and of finite instances, respectively.

Despite its basic nature, the boundedness problem is known to be undecidable for even very rudimentary classes of programs – a fact which frustrated all hopes to systematically eliminate bounded, i.e. spurious, recursion in effective tools for query optimization. See for instance [24] for the undecidability of (f-)boundedness for Datalog programs with binary IDB predicates, as well as for Datalog programs with just monadic IDB predicates but with EDB negation or even just with inequalities in the bodies. One of the few major decidability results is the following from [17].

Theorem 8.2 (Cosmadakis,Gaifman,Kanellakis,Vardi) $\text{BDD}_f(\mathcal{P}) = \text{BDD}_c(\mathcal{P})$ *is decidable for the class $\mathcal{P}$ of all monadic Datalog programs.*

The following result from classical model theory is of fundamental importance for links between boundedness and first-order (FO) definability. It speaks about IDB-positive programs Π that are first-order in the sense that the bodies of rules can be expressed in FO, by formulas that are positive in all IDB predicates (which guarantees monotonicity). We use the term *first-order programs* in this sense. We say that the fixed point of Π is FO-definable over the class $\mathcal{I}$

if each IDB predicate in the least fixed point $\Pi^\infty(I)$ is definable in terms of the EDB predicates by some first-order formula, uniformly across all $I \in \mathcal{I}$.

Theorem 8.3 (Barwise–Moschovakis [8]) *An IDB-positive first-order program* Π *is bounded in the classical sense if, and only if, the fixed point of* Π *is* FO*-definable over the class of all (finite and infinite) instances.*

Analogous equivalences can be derived for many natural fragments $\mathrm{L} \subseteq \mathrm{FO}$, where boundedness of IDB-positive L-programs is equated with L-definability of their fixed points. This is true in particular also for the guarded negation fragment $\mathrm{GNFO} \subseteq \mathrm{FO}$.

Moreover, for many well-behaved fragments $\mathrm{L} \subseteq \mathrm{FO}$ there are model theoretic transfer results that say that an L-program Π is bounded over $\mathcal{I}$ if, and only if, it is bounded over some subclass $\mathcal{I}_0 \subseteq \mathcal{I}$. A case of particular interest is a *finite model property* for boundedness, which links the classical notion to its finite model theory version. This, too, is available in the case of GNFO.

Definition 8.4 A GNFO-program is an IDB-positive program Π with rules of the form

$$X\mathbf{x}_s \;\leftarrow\; \alpha_s(\mathbf{x}_s) \wedge \phi_s(\mathbf{X}, \mathbf{x}_s)$$

where $\phi_s \in$ GNFO is positive in the IDB predicates $\mathbf{X}$ and α_s is an EDB atom guarding the variable tuple $\mathbf{x}_s$ in the head.

The following say that for GNFO we are in the ideal situation that f-boundedness and c-boundeness coincide, and that the classical and finite model theory variants of the Barwise–Moschovakis correspondence hold. The finite model theory analogue is the least straightforward of these.[3]

Proposition 8.5 *For* GNFO*-programs* Π *and their least fixed points* Π^∞*, t.f.a.e.:*

(i) Π^∞ *is* FO*-definable over all finite instances.*
(ii) Π^∞ *is* FO*-definable over all unrestricted instances.*
(iii) Π^∞ *is* GNFO*-definable over all finite instances.*
(iv) Π^∞ *is* GNFO*-definable over all unrestricted instances.*
(v) Π *is bounded over all finite instances.*
(vi) Π *is bounded over all unrestricted instances.*

Another crucial transfer property for BDD(GNFO) is based on the notion of *treewidth*. In [7], it was suggested that the key to the good computational behavior of GNFO and GNFP lies in the fact that these logics have a *tree-like model property*: for testing the satisfiability and the entailment of formulas, it suffices to consider structures of bounded treewidth. The same notion provides the key to decidability of boundedness as well.

The *width* $\mathrm{w}(\Pi)$ of a GNFO-program Π is the maximum number of element variables used in any of its rules in DNF.

[3]It is known, for instance, that the universal fragment of FO, despite its finite model property, does not satisfy this analogue: there is a purely universal program whose limit is uniformly definable in universal FO across all finite instances, although it is unbounded over finite instances.

Lemma 8.6 *A* GNFO*-program* Π *of width* $\leq w$ *is bounded over all unrestricted instances if, and only if, it is bounded over the class of all (possibly infinite) instances of treewidth at most* w.

Proof. Each finite stage $\mathbf{X}^n$ of Π can be defined by a sequence of GNFO-formulas whose width is bounded by w. In particular, for each natural number n, boundedness of Π at stage $n \geq 1$, w.r.t. a class of structures, is equivalent to the validity of a certain GNFO-sentence of width w, on that class of structures. Since a GNFO-sentence of width w is valid on arbitrary structures if and only if it is valid on structures of treewidth at most w (cf. [7, 33]), the claim follows. $\square$

We turn to decidability of BDD_c(GNFO) and of *full boundedness* (to be defined below) of GN-Datalog. Given the meager history of decidability results concerning boundedness for database purposes, it is interesting that here is one considerable extension of *the* early decidability result for monadic Datalog from [17], cf. Theorem 8.2 above.

We note that GN-Datalog is stricly more expressive than monadic Datalog, but avoids the dangers of negation that render boundedness undecidable, for instance, in the extension of monadic Datalog by just inequalities, or by negative as well as positive access to some binary EDB predicates.

Technically, the following decidability assertion is an easy corollary to the decidability results for monadic second-order logic and guarded second-order logic over tree-like structures in [10]. These results in turn are based on a non-trivial reduction to an automata theoretic decidability result of Colcombet and Löding, which, in the relevant strength needed here, has not been published yet. As in [10] we indicate this caveat formally as an assumption (ILT), which refers to the decidability of *limitedness* for weighted parity automata on infinite trees, as announced in connection with progress on earlier work in [16].

Recall that BDD_c(GNFO) and BDD_f(GNFO) coincide.

Theorem 8.7 (assuming ILT) *Boundedness for* GNFO*-programs is decidable.*

Proof. The GNFO-formulas in an GNFO-program can be translated into explicitly guarded formulas of guarded second-order logic GSO (denoted GSO^* in [10]). The result then follows from the decidability of $\mathrm{BDD}(\mathrm{GSO}^*, \mathcal{W}_k)$, boundedness for GSO^* over the class of structures of treewidth k, where both the GSO^*-formulas and the parameter k are treated as input (Theorem 8.8 in [10]). We apply this to the GSO^*-translation of the input GNFO-program Π over the class $\mathcal{W}_k$ for $k := \mathrm{w}(\Pi)$.[4] By Lemma 8.6, (ii), this is a valid reduction. $\square$

Towards our interest in GN-Datalog, with its stratified use of guarded negation as defined in Section 4, we extend the notion of boundedness from Definition 8.1 as follows.

Definition 8.8 A GN-Datalog program $\tilde{\Pi} = (\Pi_i)_{i \leq n}$ is called *fully f/c-bounded* over a class of instances $\mathcal{I}$ if each stratum Π_i is f/c-bounded over the class of all instances

[4]NB: since Π really corresponds to a *system* of least fixed points in several IDB predicates X, we need the result for systems of simultaneous fixed points in GSO^* from [10], as discussed in the proof sketch for Theorem 11.5 there.

obtained from instances in $\mathcal{I}$ by evaluating all IDB predicates from lower strata according to $\tilde{\Pi}_{<i}$ and treating them as EDB for Π_i. Equivalently, a GN-Datalog program $\tilde{\Pi} = (\Pi_i)_{i \leq t}$ is fully f/c-bounded if there are natural numbers $k_1, \ldots, k_t$ such that for all finite/unrestricted instances I, $\Pi^\infty(I) = \Pi_t^{k_t}(\Pi_{t-1}^{k_{t-1}}(\cdots \Pi_1^{k_1}(I)\cdots))$.

Corollary 8.9 (assuming ILT) *For* GN-*Datalog, full f-boundedness is decidable and coincides with full c-boundedness.*

Proof. The proof is by induction on the number of strata. Note that by definition of full boundedness, a stratified GN-Datalog program $\tilde{\Pi} = (\Pi_i)_{i \leq n}$ fails to be fully bounded if, and only if, there is a least stratum $m \leq n$ such that Π_m is unbounded over the class of instances obtained by evaluating all IDB predicates of lower strata according to $\tilde{\Pi}_{<m}$. Since these lower strata are bounded, this partial evaluation is in fact GNFO-definable. It follows that the above arguments concerning the GNFO-variant of the Barwise–Moschovakis theorem and its finite model theory version carry through – stratum by stratum, and up to the first stratum that turns out to be unbounded, if any. This also reduces the decidability claim to that in Theorem 8.7. $\square$

We remark that the passage through boundedness for GSO* over $\mathcal{W}_k$, which is known to be of non-elementary complexity even for $k = 1$, prevents us from extracting any reasonable complexity bounds. It is conceivable, of course, that alternative methods yield such bounds (as is the case for other special cases of interest, besides that of monadic Datalog, that also follow from the master result of [10]).

9. DISCUSSION

9.1 Further extensions of GN-SQL

Inequalities. GN-SQL can be viewed as a well-behaved query language extending unions of conjunctive queries with a restricted form of negation. In this sense, it is natural to compare GN-SQL to UCQ($\neq$), the language of unions of conjunctive queries with inequalities. Like GN-SQL, UCQ($\neq$) is computationally well-behaved: query containment is Π_2^p-complete [27, 38], the combined complexity of query evaluation is NP-complete, and the data complexity of open world query answering is NP-complete w.r.t. a large class of constraints [19], cf. also [30]. In this light, and in the light of Figure 3, the question arises whether we can extend GN-SQL to allow for the use of (unguarded) inequalities.

Let us denote by GN-SQL($\neq$) the extension of GN-SQL where conditions may make use of the inequality relation ($\neq$), but the inequality relation cannot be used to guard negations. It is easy to see that Theorem 7.1 extends to GN-SQL($\neq$) — we may view the inequality as just another relation that is part of the input instance. All the other results we obtained for GN-SQL, however, fail for GN-SQL($\neq$). This follows from the fact that it is possible to express functional dependencies in GN-SQL($\neq$). Indeed, every functional dependency

$$\forall \mathbf{xyz}uv(F(\mathbf{x},\mathbf{y},u) \wedge F(\mathbf{x},\mathbf{z},v) \rightarrow u = v)$$

is equivalent to the GNFO sentence with inequality

$$\neg\exists \mathbf{xyz}, u, v(F(\mathbf{x},\mathbf{y},u) \wedge F(\mathbf{x},\mathbf{z},v) \wedge u \neq v) \ ,$$

which can easily be expressed in GN-SQL($\neq$) as well. Recall that inclusion dependencies too can be expressed in GN-SQL. This, together with classical results in dependency theory (cf. [1]) and Theorem 7.8(ii)), implies the following:

Theorem 9.1 *(i) GN-SQL($\neq$) is not finitely controllable for satisfiability or query containment.*

(ii) The satisfiability and query containment problems for GN-SQL($\neq$) are undecidable (both on finite instances and on unrestricted instances).

(iii) There is a GN-SQL($\neq$) query for which open world query answering is undecidable.

Known results for various description logics contained in GNFO imply that OWA answering for GNFO($\neq$) queries is undecidable when the query is part of the input [34]. Theorem 9.1 strengthens this by showing that the problem is undecidable already for a fixed GNFO($\neq$) query. Naturally, similar results can be obtained for the extension of GN-Datalog with inequalities.

Constants and Comparisons. GN-SQL queries, as we defined them, cannot contain constant values, nor arithmetical comparisons (i.e., conditions of the form $t_1 < t_2$). Indeed, over linearly ordered domains, inequalities can be expressed using arithmetical comparisons ($x \neq y$ is equivalent to $x < y \vee y < x$), and hence, by Theorem 9.1, most problems immediately become undecidable when arithmetical comparisons are allowed. However, as we will show, our results *do* generalize to the extension of GN-SQL where (i) queries may contain constant values, and (ii) arithmetical comparisons of the form $t_1 < t_2$ are allowed provided that at least one of t_1, t_2 is a constant value.

In what follows, let LIN $= (D, \prec)$ be any ordered domain (where D is a countable set and $\prec$ is a total order on D) that is “reasonable” in the sense that the following problems are all solvable in polynomial time (for some appropriate representation of the elements of D):

1. given $d_1, d_2 \in D$, is it the case that $d_1 \prec d_2$?
2. given $d \in D$, does there exist $d' \in D$ with $d' \prec d$?
3. given $d \in D$, does there exist $d' \in D$ with $d \prec d'$?
4. given $d_1, d_2 \in D$, does there exist $d' \in D$ with $d_1 \prec d' \prec d_2$?

Essentially, all the usual ordered domains, such as the natural numbers $(\mathbb{N}, <)$, the rational numbers $(\mathbb{Q}, <)$, and the strings $(A^*, <_{lex})$ over a finite ordered alphabet A, are reasonable in this sense.

Let GN-SQL(LIN) be the extension of the GN-SQL syntax where (i) all terms t are allowed to be either of the form R.ATTR (as before) or to be an element of LIN (in which case we call t a *constant*); and (ii) for all terms t_1, t_2 of which at least one is a constant value, $t_1 < t_2$ is allowed as an atomic condition. The semantics of GN-SQL(LIN) queries is only well-defined for instances whose active domain is a subset of LIN. Therefore, we restrict attention to such instances.

All results for GN-SQL that we have presented can be extended to GN-SQL. For simplicity, we sketch the relevant construction here only for the query containment problem.

Theorem 9.2 *Let* LIN *be any reasonable ordered domain. GN-SQL(*LIN*) query containment is 2ExpTime-complete.*

Aggregation. Recall that GN-SQL does not allow for any form of aggregation that is available in SQL. This is for good reason: allowing even simple forms of aggregation such as counting would quickly lead to undecidability, since query containment for unions of conjunctive queries under the bag semantics is undecidable [25].

9.2 Further Extensions of GN-Datalog

Allowing IDBs As Guards. If in the definition of negation-guarded Datalog rules one permits also the use of IDB atoms *from the same or lower stratum* as guards, this can result in an exponential gain in succinctness but does not increase the expressive power. (A simple induction on strata and on stages of inductive definitions of IDB predicates confirms that all tuples added to the interpretation of IDB predicates are guarded by some EDB atom.) Query evaluation complexity, however, suffers an exponential blow-up as a consequence of this relaxation.

Proposition 9.3 (GN-Datalog with IDB guards) *Answering GN-Datalog queries with IDB atoms allowed as guards is ExpTime-complete in combined complexity.*

Capturing the Alternation-Free Fragment of GNFP. In [20], an extension of Datalog-LIT was presented, called Datalog-LITE, which includes "generalized literals" and was shown to capture the alternation-free fragment of guarded fixed point logic (GFP). We expect that GN-Datalog can be similarly extended, in order to subsume Datalog-LITE and capture the alternation-free fragment of GNFP.

10. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] A. Arnold and D. Niwinski. *Rudiments of μ-calculus.* Elsevier, 2001.

[3] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Art. Intell.*, 175(10):1620–1654, 2011.

[4] J.-F. Baget, M.-L. Mugnier, S. Rudolph, and M. Thomazo. Complexity Boundaries for Generalized Guarded Existential Rules. Technical Report lirmm-00568935, LIRMM, 2011.

[5] J.-F. Baget, M.-L. Mugnier, S. Rudolph, and M. Thomazo. Walking the complexity lines for generalized guarded existential rules. In *Proc. IJCAI*, pages 712–717, 2011.

[6] V. Bárány and M. Bojanczyk. Finite satisfiability for guarded fixpoint logic. *CoRR*, abs/1104.2262, 2011.

[7] V. Bárány, B. ten Cate, and L. Segoufin. Guarded negation. In *Proc. ICALP*, 2011.

[8] J. Barwise and Y. N. Moschovakis. Global inductive definability. *J. Symb. Log.*, 43(3):521–534, 1978.

[9] H. Björklund, W. Martens, and T. Schwentick. Optimizing conjunctive queries over trees using schema information. In *Proc. MFCS*, 2008.

[10] A. Blumensath, M. Otto, and M. Weyer. Decidability results for the boundedness problem. Preprint, 2011.

[11] A. Calì, G. Gottlob, and M. Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In *Proc. KR*, 2008.

[12] A. Calì, G. Gottlob, and T. Lukasiewicz. Tractable query answering over ontologies with Datalog$^{\pm}$. In *Proc. Description Logics Workshop*, 2009.

[13] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. KR*, pages 260–270, 2006.

[14] B. ten Cate and L. Segoufin. Unary negation. In *Proc. STACS*, pages 1–35, 2011.

[15] S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive datalog programs. *J. Comput. Syst. Sci.*, 54(1):61–78, 1997.

[16] T. Colcombet and C. Löding. The nesting-depth of disjunctive μ-calculus for tree languages and the limitedness problem. In *Proc. CSL*, 2008.

[17] S. Cosmadakis, H. Gaifman, P. Kanellakis, and M. Vardi. Decidable optimization problems for database logic programs. In *Proc. STOC 1988*, pages 477–490, 1988.

[18] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[19] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[20] G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: a deductive query language with linear time model checking. *ACM Trans. Comput. Log.*, 3(1):42–79, 2002.

[21] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for Web information extraction. *J. ACM*, 51(1):74–113, 2004.

[22] E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.

[23] E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *LICS*, pages 45–54. IEEE Comp. Soc., 1999.

[24] G. Hillebrand, P. Kanellakis, H. Mairson, and M. Vardi. Undecidable boundedness problems for datalog programs. *J. of Logic Prog.*, 25:163–190, 1995.

[25] Y. E. Ioannidis and R. Ramakrishnan. Containment of conjunctive queries: beyond relations as sets. *ACM Trans. Database Syst.*, 20:288–324, 1995.

[26] D. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comp. Syst. Sci.*, 28:167–189, 1984.

[27] A. Klug. On conjunctive queries containing inequalities. *J. ACM*, 35:146–160, January 1988.

[28] D. Leinders, M. Marx, J. Tyszkiewicz, and J. Bussche. The semijoin algebra and the guarded fragment. *J. of Logic, Lang. and Inf.*, 14:331–343, June 2005.

[29] L. Libkin. Expressive power of SQL. *Theor. Comput. Sci.*, 296(3):379–404, 2003.

[30] A. Madry. Data exchange: On the complexity of answering queries with inequalities. *Inf. Process. Lett.*, 94(6):253–257, 2005.

[31] A. Nash, A. Deutsch, and J. Remmel. Data exchange, data integration and chase. Technical Report CS2006-0859, UCSD, 2006.

[32] M. Ortiz, D. Calvanese, and T. Eiter. Data complexity of Answering Unions of Conjunctive Queries in SHIQ. In *Proc. Description Logics Workshop*, 2006.

[33] M. Otto. Expressive completeness through logically tractable models. Submitted, 2012.

[34] R. Rosati. The limits of querying ontologies. In *Proc. ICDT*, pages 164–178, 2006.

[35] Sloan Digital Sky Survey. Sky server sample SQL queries. `http://skyserver.sdss.org/public/en/help/docs/realquery.asp`. Accessed Sept. 25, 2011.

[36] TPPC. TPC-DS benchmark. `http://www.tpc.org/tpcds/`. Release 2008-01-21.

[37] TPPC. TPC-H benchmark. `http://www.tpc.org/tpch/`. Release 2.14.3.

[38] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *J. Comput. Syst. Sci.*, 54(1):113–135, 1997.

[39] M. Y. Vardi. The complexity of relational query languages. In *Proc. STOC 1982*, pages 137–146, 1982.

[40] K. Wagner. More Complicated Questions about Maxima and Minima, and some Closures of NP. *Theoretical Computer Science*, 51(1-2):53–80, 1987.

APPENDIX

A. MISSING PROOFS

A.1 Proof of Theorem 2.2 and inexpressibility claims

Proof. From GN-RA expressions to GNFO formulas, there is a straightforward inductive linear translation. More precisely, the following table describes how to translate each GN-RA expression E of arity k into an equivalent (therefore domain-independent) GNFO formula $\varphi_E(x_1,\ldots,x_k)$.

R	$R(x_1,\ldots,x_k)$
$\sigma_{i=j}(E)$	$\varphi_E(\mathbf{x}) \wedge x_i = x_j$
$\pi_{i_1\ldots i_m}(E)$	$\exists \mathbf{z} \varphi_E(\mathbf{z}) \wedge \bigwedge_{j=1}^m x_j = z_{i_j}$
$E \times E'$	$\varphi_E(x_1,\ldots,x_{k_1}) \wedge \varphi_E(x_{k_1+1},\ldots,x_{k_1+k_2})$
$E \cap E'$	$\varphi_E(x_1,\ldots,x_k) \wedge \varphi_E(x_1,\ldots,x_k)$
$E \cup E'$	$\varphi_E(x_1,\ldots,x_k) \vee \varphi_E(x_1,\ldots,x_k)$
$\pi_{i_1\ldots i_m}(R) \setminus E$	$\exists \mathbf{z}\big(R(\mathbf{z}) \wedge \neg\varphi_E(\mathbf{x}) \wedge \bigwedge_{j=1}^m x_j = z_{i_j}\big)$

For the converse direction, we proceed as follows: we first construct a GN-RA expression ADOM that defines the active domain of the instance (the union of all unary projections of atomic relations). Next, for each sequence of variables $\mathbf{x} = x_1,\ldots,x_k$ and for each atomic GNFO formula ϕ whose free variables are included in $\mathbf{x}$, we compute a k-ary GN-RA expression $\mathsf{tr}_{\mathbf{x}}(\phi)$ that is equivalent to it under the active domain semantics (i.e., over structures whose domain coincides with the active domain). For instance $\mathsf{tr}_{x_1,x_2,x_3}(R(x_2,x_2)) = \pi_{1,2,4}\sigma_{2=3}(\text{ADOM} \times R \times \text{ADOM})$, and $\mathsf{tr}_{x_1,x_2,x_3}(x_1 = x_2) = \pi_{1,1,2}(\text{ADOM} \times \text{ADOM})$. Finally, the translation $\mathsf{tr}_{\mathbf{x}}(\cdot)$ is extended to complex GNFO formulas. Conjunction, disjunction and existential quantification are translated as intersection, union, and projection. Hence, the only remaining case is for $\mathsf{tr}_{\mathbf{x}}(\alpha(\mathbf{y}) \wedge \neg\phi(\mathbf{y}))$, where the variables in $\mathbf{y}$ are included in the variables in $\mathbf{x}$. If the guard α is a relational atom, $\mathsf{tr}_{\mathbf{x}}(\alpha(\mathbf{y}) \wedge \neg\phi(\mathbf{y}))$ can be defined as the GN-RA expression obtained from $\mathsf{tr}_{\mathbf{y}}(\alpha) - \mathsf{tr}_{\mathbf{y}}(\phi)$ by (i) pulling out selections and projections as necessary in order to turn the first argument of the complementation operator into a projection of an atomic relation; and (ii) taking a product with ADOM for each variable from $\mathbf{x}$ that is not included in $\mathbf{y}$.

If the guard α is of the form $y_1 = y_2$, then $\mathsf{tr}_{\mathbf{y}}(\alpha(\mathbf{y}) \wedge \neg\phi(\mathbf{y}))$ is defined, in the first instance, as $\pi_{1,1}(\text{ADOM} - \pi_1\sigma_{1=2}\mathsf{tr}_{\mathbf{y}}(\phi))$. Since ADOM is in general a union of all unary projections of atomic relations, we need to pull out the union from the scope of the complementation operator. This is where an exponential blow-up may be incurred. □

Proposition A.1 *The following RA expressions are not equivalent to GN-RA expressions:*

1. $(\pi_1(R) \times S) - \pi_{1,1}(R)$
2. $\pi_{1,4}(\sigma_{2=3}(R \times R)) - R$
3. $\pi_1(R) - \pi_1((\pi_1(R) \times S) - R)$

Proof. In [7], the notion of *GN-bisimulation* was introduced, and it was shown that GN-bisimulations preserve the truth of GNFO sentences. Together with Theorem 2.2 this allows us to show non-expressibility of the above RA expressions in GN-RA. Note that if any of the above RA expressions was definable in GN-RA, then also its boolean projection would be definable in GN-RA. It can be shown that

1. the instance $\{R(a,b), S(a), S(c)\}$, which satisfies the boolean projection of $(\pi_1(R) \times S) - \pi_{1,1}(R)$, is GN-bisimilar to the instance $\{R(a,b), S(a)\}$, which does not.
2. the instance $\{R(a,b), R(b,c), R(a,c), R(b,d)\}$, which satisfies the boolean projection of $\pi_{1,4}(\sigma_{2=3}(R \times R)) - R$ is GN-bisimilar to the instance $\{R(a,b), R(a,c), R(b,c)\}$, which does not.
3. the instance $\{R(a,b), S(a), S(c)\}$, which satisfies the boolean projection of $(\pi_1(R) \times S) - \pi_{1,1}(R)$, is GN-bisimilar to the instance $\{R(a,b), S(a)\}$, which does not. □

A.2 Proof of Theorem 3.1

Proof sketch. Let q be any (closed) GN-SQL query. We may assume without loss of generality that each tuple variable R occurring in q is declared in exactly one from-clause, and therefore has a unique associated relation name, that we will denote by REL_R. By a simultaneous induction, we can

- translate each (open or closed) GN-SQL query q to a GNFO formula $\phi_q(\mathbf{x})$, where $\mathbf{x}$ is a sequence of first-order variables, one for each attribute name belonging to the type of the query q and one for each term $R.\text{ATTR}$ where R is a tuple variable that occurs freely in q and ATTR is an attribute name belonging to the type of REL_R,
- translate each GN-SQL condition c to a GNFO formula $\phi_c(\mathbf{x})$, where $\mathbf{x}$ is a sequence of first-order variables, one for each term $R.\text{ATTR}$ where R is a tuple variable that occurs freely in c and ATTR is an attribute name belonging to the type of REL_R.

We omit the detailed definition of the translation, which is straightforward. The clause for not is as follows:

$$\phi_{\mathsf{not}(condition)}(\mathbf{x}) = \text{REL}_R(\mathbf{x}) \wedge \neg\phi_{condition}(\mathbf{x})$$

It is not hard to see that each closed GN-SQL query q is equivalent to its GNFO translation ϕ_q. In particular, this implies that ϕ_q is domain independent.

For the converse translation, from domain-independent GNFO formulas to GN-SQL queries, it is convenient to first assume that we have at our disposal a relation ADOM with a single attribute A containing all elements belonging to the active domain. As we will show, using such a relation, it is quite straightforward to give an inductive polynomial translation from GNFO to GN-SQL. On the other hand, all usage of ADOM can be eliminated at the cost of an exponential blow-up. To see this, recall that relation names can only appear in FO-SQL queries in the from-clause of a select-from-where expression. Thus, any occurrence of ADOM must be of the form

$$\textsf{select } \alpha \textsf{ from } (\ldots, \textsc{adom}\ R, \ldots) \textsf{ where } \beta$$

where, in addition, the expressions α and β may refer to $R.A$. We may equivalently replace such an expression by the union of all expressions of the following form (for all relation names $\textsc{rel}_i$ and attribute names $\textsc{attr}_j$):

$$\textsf{select } \alpha' \textsf{ from } (\ldots, \textsc{rel}_i\ R, \ldots) \textsf{ where } \beta'$$

where α' and β' are obtained from α and β by replacing all occurrences of $R.A$ by $R.\textsc{attr}_j$. Clearly, applying this transformation for all occurrences of ADOM yields an equivalent query that does not make use of ADOM and that is at most singly exponentially larger than the original query.

Next, we explain how to translate domain-independent GNFO formulas to GN-SQL queries with the help of the ADOM relation. Let $\phi(\mathbf{x})$ be any domain-independent GNFO formula. We assume w.l.o.g. that ϕ does not reuse any variables, and associate to each first-order variable z a corresponding distinct tuple variable R_z (whose type, in the expressions below, will consist of a single attribute named A). Next, we inductively translate each GNFO formula ϕ to a GN-SQL condition ϕ^*, as follows.

$$\begin{array}{ll}
(x = y)^* & = (R_x.A = R_y.A) \\
\textsc{rel}(x_1,\ldots,x_n) & = \textsf{exists(select } R.A_1 \textsf{ from } \textsc{rel}\ R \textsf{ where} \\
 & \quad R.A_1 = R_{x_1}.A \textsf{ and } \ldots\ R.A_n = R_{x_n}.A) \\
(\phi \wedge \psi)^* & = \phi^* \wedge \psi^* \\
(\phi \vee \psi)^* & = \phi^* \vee \psi^* \\
(\exists x\ \phi)^* & = \textsf{exists(select } R_x.A \textsf{ from } \textsc{adom}\ R_x \textsf{ where } \phi^*) \\
(\textsc{rel}(\mathbf{x}) \wedge \neg\phi)^* & = \textsf{exists(select } R.A_1 \textsf{ from } \textsc{rel}\ R \textsf{ where} \\
\multicolumn{2}{r}{R.A_1 = R_{x_1}.A \textsf{ and } \ldots \textsf{ and } R.A_n = R_{x_n}.A \textsf{ and not}(\widehat{\phi^*}))} \\
(x = y \wedge \neg\phi)^* & = (R_x.A = R_y.A) \textsf{ and not } \phi[x/y]^*
\end{array}$$

where, in the second clause and in the 6th clause, the schema of the relation REL is $\{A_1, \ldots, A_n\}$, and where, in the 6th clause, $\widehat{\phi^*}$ is obtained from ϕ^* by replacing each term $R_{x_i}.A$ by $R.A_i$. In the last clause, $\phi[x/y]$ is the formula obtained from ϕ by replacing each free occurrence of x by y, so that the formula in question has only one free first-order variable.

Finally, starting with a GNFO formula $\phi(x_1, \ldots, x_n)$ we define the query q_ϕ as follows:

$$\begin{array}{l}
\textsf{select } R_1.A \textsf{ as } \textsc{attr}_1, \ldots, R_n.A \textsf{ as } \textsc{attr}_n \\
\textsf{from } \textsc{adom}\ R_1, \ldots, \textsc{adom}\ R_n \textsf{ where } \phi^*
\end{array}$$

where $\textsc{attr}_1, \ldots, \textsc{attr}_n$ are distinct attribute names. It is easy to show that each domain-independent GNFO formula is equivalent to the GN-SQL query obtained from it in the above way. □

A.3 Proof of Theorem 4.4

PROOF. Consider any non-recursive GN-Datalog query $q = (\tilde{\Pi}, Ans)$ with $\tilde{\Pi} = (\Pi_1, \ldots, \Pi_n)$. A straightforward induction on k shows that, for every $k \leq n$ and for every $X \in \mathrm{IDB}^{\Pi_k}$, there is a GNFO formula ϕ that defines the relation computed by X. The GNFO formula in question can be obtained by taking the disjunction of all bodies of rules that have X in the head, replacing all occurrences of IDBs $Y \in \mathrm{IDB}^{\Pi_\ell}$ with $\ell < k$ by their (previously obtained) defining GNFO formulas. Finally, by taking the query Ans and replacing each IDB $X \in \mathrm{IDB}^{\Pi_n}$ by its defining GNFO formula, we obtain a GNFO formula that is equivalent to q, and, in particular, domain independent, since q is domain independent.

Conversely, let $\phi(\mathbf{x})$ be a domain-independent GNFO formula. We may assume that $\phi(\mathbf{x})$ is in DNF. This may require an exponential blow-up. First, we construct a GN-Datalog program with a unary IDB ADOM that computes the active domain, as well as a binary IDB $X_=$ that computes the relation $\{(x, x) \mid x \in \textsc{adom}\}$, which will be used for translating equality statements. We omit the construction, which is straightforward. Next, by induction, we construct for each subformula of ϕ that is of the form $\alpha \wedge \neg\chi$ a non-recursive GN-Datalog program with IDB relations $X_{\alpha\wedge\chi}$ and $X_{\alpha\wedge\neg\chi}$ computing the relation defined by $\alpha \wedge \chi$ and $\alpha \wedge \neg\chi$ under the active-domain semantics (i.e., on structures whose active domain is the entire domain). In particular, if α is a relational atom, then the program includes the rule

$$X_{\alpha\wedge\neg\chi}(\mathbf{x}) \leftarrow \alpha(\mathbf{x}) \wedge \neg X_{\alpha\wedge\chi}(\mathbf{x})$$

If α is an equality statement, we proceed similarly, using the $X_=$ IDB relation introduced above as a guard.

Finally, if $\phi(\mathbf{x})$ is of the form $\phi_1(\mathbf{x}_1) \vee \ldots \vee \phi_n(\mathbf{x}_n)$, we define Ans to be the union of the conjunctive queries $Ans_i(\mathbf{x}) := (\, X_{\phi_i}(\mathbf{x}_i) \wedge \bigwedge_{x\in\mathbf{x}} \textsc{adom}(x)\,)$. From the domain-independence of $\phi(\mathbf{x})$, we obtain that $q = (\tilde{\Pi}, Ans)$ is equivalent to ϕ. □

A.4 Proof of Theorem 6.1

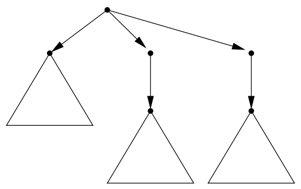
It was shown in [14] that satisfiability for UNFO is 2ExpTime-hard, where UNFO is a syntactic fragment of GNFO. As we explain below, the construction can be adapted to prove the same lower bound result already for GNFO formulas in DNF. In addition, we can easily ensure that the GNFO formulas in question are domain independent, and force the existence of a fixed unary predicate denoting the active domain. Under these restrictions, the translation from GNFO to GN-SQL (Theorem 3.1) and the translation from GNFO to non-recursive GN-Datalog (heorem 4.4) both runs in polynomial time. Hence, we obtain 2ExpTime-hardness for satisfiability, and therefore also for query containment, for GN-SQL and non-recursive GN-Datalog.

Proposition A.2 *There is a fixed schema such that the satisfiability problem for GNFO formulas in DNF is 2ExpTime-hard, both on arbitrary structures and on finite structures.*

PROOF SKETCH. The same result, without the DNF requirement, was shown in [14], in the context of a fragment of GNFO called UNFO. We briefly sketch the construction used in the proof in [14], and explain how it can be adapted to use only GNFO formulas in DNF.

Fix an alternating 2^n-space bounded Turing machine M whose word problem is 2ExpTime-hard. Let w be a word in the input alphabet of M. We construct a formula ϕ_w that is satisfiable if and only if M accepts w. Moreover, if ϕ_w is satisfiable, then in fact it is satisfied in some finite tree structure. In this way, we show that the lower bound holds not only for arbitrary structures, but also for finite trees and for any class in-between. The formula ϕ_w describes an (alternating) run of M starting in the initial state with w on the tape, and ending in a final configuration.

The run is encoded as a big tree whose nodes correspond to configurations, whose child relation correspond to successive configurations, and where each node has in addition a small subtree of height n attached to it, that is used to describe the tape content at that configuration. Here is an illustration of a configuration with two successor configurations (but we allow more than two successor configurations):



Each small subtree has depth exactly n. The internal nodes of the subtree are label with a unary predicate P. Hence a path from its root to one of its leaf correspond to a bit string of length n denoting a position of the tape. The label of the leaf codes the content of the tape at that position.

The formula first enforces that the small subtrees have the desired structure and that all positions are realized in at least one leaf of each small subtree. Since we don't have inequality, we cannot force that it is realized exactly once, but we can force in GNFO that all nodes where it is realized satisfy the same relevant unary predicates A:

$$\neg\exists xy(\text{leaf}(x) \wedge \text{leaf}(y) \wedge x \uparrow^n\downarrow^n y \wedge \bigwedge_i (P_i(x) \leftrightarrow P_i(y)) \wedge A(x) \wedge \neg A(y))$$

Here, $x \uparrow^n\downarrow^n y$ is a short for the GNFO formula describing the fact that there is a path of the form $\uparrow^n\downarrow^n$ from x to y, leaf(x) is a short for $\neg\exists y Rxy$, and $P_i(x)$ is a shortcut for $\exists y(x \uparrow^{n-i} y \wedge P(y))$.

The following formula suc(x, y) expresses that x and y denote the same tape position in successive configurations, and it uses only unary negation:

$$\text{suc}(x, y) := \text{leaf}(x) \wedge \text{leaf}(y) \wedge (x \uparrow^{n+1}\downarrow^{n+2} y) \wedge \bigwedge_i (P_i(x) \leftrightarrow P_i(y))$$

Note that the first half of the formula says that x and y are tape cells of successive configurations. Using this formula, we can specify all relevant properties of the run (the encoding will involve formulas of the form $\forall x(\text{leaf}(x) \wedge \phi(x) \rightarrow \exists y(\text{suc}(x, y) \wedge \psi(y))))$.

This concludes the outline of the construction used in [14] for showing 2ExpTime-hardness of the satisfiability problem for (a fragment of) GNFO.

The above proof clearly uses GNFO formulas that are not in DNF, and the straightforward way to bring the formulas in DNF, by "pulling out disjunction", would lead to formulas whose length is exponential in n. The problem, here, lies in the formula $\bigwedge_i(P_i(x) \leftrightarrow P_i(y))$ expressing that two leaf nodes, in the same configuration or in successor configurations, encode the same memory location. This use of disjunction can be avoided using a construction from [9]. In particular, we enrich our encoding of Turing machine configurations as follows: to each node x of the structure, we attach a small substructure consisting of nodes that we mark with a fresh unary predicate Q in order to distinguish them from the nodes that belong to the "main structure" (i.e., the structure as it was before adding all these new small substructures). The exact substructure that we attach to a node x depends on whether or not the node satisfies P. If a node x it satisfies P, we create a new node y and add edges $E(x, y)$ and $R(x, y)$. If, on the other hand, x does not satisfy P, we create new nodes y and z and add edges $E(x, z), R(x, y), R(y, z)$. Here, E is a new binary predicate. This modification of the structure has the consequence that we can avoid the use of disjunction in comparing whether two leaf node encode the same memory location: suppose that x and y be leaf nodes of the same configuration subtree. Then $(P_i(x) \leftrightarrow P_i(y))$ can be equivalently expressed as

$$\exists uu'vv'(x \uparrow^{n-i} u \wedge E(u, u') \wedge y \uparrow^{n-1} v \wedge E(v, v') \wedge u' \uparrow^{i+2}\downarrow^{i+2} v')$$

In a similar way, we can express, without using disjunction, the fact that two nodes encode the same memory location in *successive* configuration subtrees. We omit the details. □

A.5 Proof of Theorem 7.2

PROOF. Upper bound: for each IDB except possibly the answer IDB, the number of tuples that may end up in the extension of the IDB is bounded by the number of tuples belonging to the extension of the EDBs, times the number of rules of the Datalog program, because each rule is guarded by an EDB (here, incidentally, what really matters for the argument is the body of each rule includes an EDB atom that contains all variables occurring in the head of the rule). Hence, the number of times a non-answer rule is applied is bounded by the number of facts in the input database instance times the number of rules of the Datalog program. Hence, the entire Datalog computation, except for the computation of the answer relation, can be viewed as a polynomial computation with an NP-oracle (for evaluating the bodies of rules). Finally, once all IDBs except the Answer IDB have been computed, we simply invoke the NP oracle once more to test if the given tuple belongs to the extension of the answer IDB.

For the lower bound we provide a reduction from the LEX(SAT) problem: given a propositional formula $\Phi(x_1, \ldots, x_n)$, determine if the value of x_n is 1 in the lexicographically least satisfying assignment, where x_n is the least significant bit. where x_n is the least significant bit. The LEX(SAT) problem is known to be P^{NP}-complete, even for 3-CNF formulas [40].

We devise a structure $\mathfrak{B}$ with a domain of two elements ($\top$ and $\bot$) endowed with a unary relation T that is true only of $\top$, a unary relation F that is true only of $\bot$, a binary relation N that holds precisely the complementary pairs $(\bot, \top)$ and $(\top, \bot)$, and a ternary relation OR that is true of all $\{\bot, \top\}$-triplets but $(\bot, \bot, \bot)$. This way, the set of satisfying assignments to every 3-clause $C(x_i, x_j, x_k)$, e.g. $x_i \vee \neg x_j \vee \neg x_k$, is the answer set to a corresponding conjunctive query $\tilde{C}(x_i, x_j, x_k)$ on $\mathfrak{B}$, such as

$\exists y_j y_k\ N(x_j, y_j) \wedge N(x_k, y_k) \wedge \mathrm{OR}(x_i, y_j, y_k)$ in this example. More generally, we can translate every 3-CNF formula $\Phi(x_1, \ldots, x_n)$ into a Datalog rule with body

$$\tilde{\Phi}(x_1, \ldots, x_n, y_1, \ldots, y_n) = \bigwedge_i N(x_i, y_i) \ \wedge \bigwedge_{C \text{ a clause}} \tilde{C}(\mathbf{x}, \mathbf{y})$$

where C ranges over the clauses of Φ.

Given a propositional formula $\Phi(x_1, \ldots, x_n)$, the idea is now to have, for each $i \leq n$, a unary IDB predicate X_i that computes the truth value of the i-th bit in the lexicographically least satisfying assignment to $\Phi(x_1, \ldots, x_n)$. These IDBs belong to different strata of the program as inductively defined by the following rules.

$$\begin{array}{rcl}
Z_1 & \leftarrow & F(x_1),\ \tilde{\Phi}(\mathbf{x}, \mathbf{y}) \\
X_1(x_1) & \leftarrow & F(x_1),\ Z_1 \\
X_1(x_1) & \leftarrow & T(x_1),\ \neg Z_1 \\
\vdots & & \\
Z_i & \leftarrow & X_1(x_1), \ldots, X_{i-1}(x_{i-1}),\ F(x_i),\ \tilde{\Phi}(\mathbf{x}, \mathbf{y}) \\
X_i(x_i) & \leftarrow & F(x_i),\ Z_i \\
X_i(x_i) & \leftarrow & T(x_i),\ \neg Z_i \\
\vdots & & \\
Ans & \leftarrow & X_n(x_n),\ T(x_n),\ \tilde{\Phi}(\mathbf{x}, \mathbf{y})
\end{array}$$

It is easy to see that the above non-recursive GN-Datalog query computes on $\mathfrak{B}$ the solution to the LEX(SAT) problem instance Φ. □

A.6 Proof of Proposition 7.3

In what follows it will be convenient to work with GNFO formulas in disjunctive normal form. Two critical dimensions of a GNFO formula in DNF are its 'width', introduced above, and its 'negation rank'. The *negation rank* $\mathrm{nrank}(\phi)$ of ϕ in DNF is the maximum number of nested negations in ϕ. Naturally, UCQs have negation rank 0. Set $\mathrm{DNF}^r_w = \{\phi \mid \phi \text{ in DNF, width}(\phi) \leq w, \mathrm{nrank}(\phi) < r\}$.

Given a structure U we let $\mathrm{atoms}(U)$ denote the set of tuples $\mathbf{u}$ forming the support of a relational atom in U. For every $\mathbf{u} \in \mathrm{atoms}(U)$ and $\mathbf{u}' \in \mathrm{atoms}(U')$ and for every $w, r \in \mathbb{N}_{\geq 0}$ let $U, \mathbf{u} \equiv^r_w U', \mathbf{u}'$ denote the fact that $U \models \psi(\mathbf{u}) \iff U' \models \psi(\mathbf{u}')$ for every $\psi \in \mathrm{DNF}^r_w$. Assuming an ambient finite relational signature, each $\equiv^r_w$ is an equivalence of finite index as there are, up to logical equivalence, only finitely many formulas in DNF^r_w. In particular, for every $\mathbf{u} \in \mathrm{atoms}(U)$ there is a formula $\chi_{U,\mathbf{u}}(\mathbf{x})$ that is a boolean combination of DNF^r_w-formulas and is characteristic of its $\equiv^r_w$-class, i.e. such that $U, \mathbf{u} \equiv^r_w U', \mathbf{u}'$ iff $U' \models \chi_{U,\mathbf{u}}(\mathbf{u}')$. We call $\chi_{U,\mathbf{u}}(\mathbf{x})$ the *DNF^r_w-type of $\mathbf{u}$ in U*. Note that $\chi_{U,\mathbf{u}}(\mathbf{x})$ is itself *not* in DNF^r_w. Also note that DNF^0_w is empty and that DNF^1_w comprises only UCQs.

Proof. For the purposes of this construction we consider the ammendment of the language of $\varphi(\mathbf{x})$ with constants $\mathbf{c}$ corresponding to the free variables $\mathbf{x}$, and regard φ as the sentence obtained from the original query by substitution of each constant c_i in place of the corresponding free variable x_i. Accordingly, given an instance I with distinguished elements $\mathbf{a}$, we treat $\mathbf{a}$ as the interpretation of the constants $\mathbf{c}$. Furthermore, we assume w.l.o.g. that ϕ is in DNF as in (1) on page and let w be the width and r the negation rank of ϕ.

For each DNF^r_w-type $\tau(\mathbf{x})$ such that $\phi \wedge \tau(\mathbf{x})$ is satisfiable we fix in advance, and independently of I, a finite model of ϕ with distinguished elements $(M^\tau, \mathbf{a}^\tau)$ realising it: $M^\tau \models \phi \wedge \tau(\mathbf{a}^\tau)$. Let C be the maximum number of facts in any of the M^τ. Note that C is independent of I.

To obtain the model M, for every $\mathbf{b} \in \mathrm{atoms}(I)$ having DNF^r_w-type $\tau(\mathbf{z})$ in J we take $(M^\mathbf{b}, \mathbf{a}^\mathbf{b})$ to be a fresh copy of $(M^\tau, \mathbf{a}^\tau)$ and attach it to I by identifying its distinguished tuple $\mathbf{a}^\mathbf{b}$ with $\mathbf{b}$ component-wise. Thus M is made up of at most Cn many facts. It remains to verify that $M \models \phi$.

Claim 1 *For every $\mathbf{b} \in \mathrm{atoms}(I)$ and $\mathbf{d} \in \mathrm{atoms}(M^\mathbf{b})$ we have $M^\mathbf{b}, \mathbf{d} \equiv^r_w M, \mathbf{d}$.*

From this claim it follows trivially that $M \equiv^r_w M^\mathbf{b}$, therefore also $M \equiv^r_w J$, since $M^\mathbf{b} \equiv^r_w J$ by choice. Because $J \models \phi$, this will allow us to conclude $M \models \phi$.

To establish Claim 1 we prove by induction on $q = 0, \ldots, r$ that $M^\mathbf{b}, \mathbf{d} \equiv^q_w M, \mathbf{d}$ for every $\mathbf{b} \in \mathrm{atoms}(I)$ and $\mathbf{d} \in \mathrm{atoms}(M^\mathbf{b})$. The latter claim is trivially true for $q = 0$. Towards the induction step assume it is true for $q-1$ and consider an arbitrary $\mathbf{b} \in \mathrm{atoms}(I)$ and $\mathbf{d} \in \mathrm{atoms}(M^\mathbf{b})$. It suffices to show that $M^\mathbf{b} \models \psi(\mathbf{d}) \iff M \models \psi(\mathbf{d})$ for all

$$\psi(\mathbf{x}) = \exists \mathbf{y} \bigwedge_l (\alpha_l(\mathbf{z}^l) \wedge \neg\psi_l(\mathbf{z}^l))$$

where each $\alpha_l(\mathbf{z}^l)$ is an atomic formula and $\psi_l \in \mathrm{DNF}^{q-1}_w$ with free variables $\mathbf{z}^l$ from among $\mathbf{xy}$. For ψ as above we additionally define $\nu(\mathbf{x}, \mathbf{y}) = \bigwedge_l (\alpha_l(\mathbf{z}^l) \wedge \neg\psi_l(\mathbf{z}^l))$.

Tackling first the easy direction, suppose that $M^\mathbf{b} \models \psi(\mathbf{d})$ and consider witnesses $\mathbf{c}$ in $M^\mathbf{b}$ such that $M^\mathbf{b} \models \nu(\mathbf{d}, \mathbf{c})$. Then $M \models \alpha_l(\mathbf{e}^l)$ is immediate for each subtuple $\mathbf{e}^l$ that relates to $\mathbf{dc}$ as $\mathbf{z}^l$ relates to $\mathbf{xy}$, while $M \models \neg\psi_l(\mathbf{e}^l)$ follows from $M^\mathbf{b} \models \neg\psi_l(\mathbf{e}^l)$ via the induction hypothesis. This proves $M \models \psi(\mathbf{d})$.

Suppose now $M \models \psi(\mathbf{d})$ and let $\mathbf{c}$ be elements of M such that $M \models \nu(\mathbf{d}, \mathbf{c})$. Our aim is to find witnesses $\mathbf{c}'$ in $M^\mathbf{b}$ such that $M^\mathbf{b} \models \nu(\mathbf{d}, \mathbf{c}')$. We distinguish two cases.

(i) If $\mathbf{c}$ lies entirely in $M^\mathbf{b}$ then, using the induction hypothesis as in the proof of the opposite direction, we can confirm that $\mathbf{c}' = \mathbf{c}$ are appropriate witnesses: $M^\mathbf{b} \models \nu(\mathbf{d}, \mathbf{c})$.

(ii) Otherwise we proceed as follows. For each atom $\alpha_l(\mathbf{z}^l)$ from ν let $\mathbf{e}^l$ be the subtuple relating to $\mathbf{dc}$ as $\mathbf{z}^l$ relates to $\mathbf{xy}$. Thus $M \models \alpha_l(\mathbf{e}^l) \wedge \neg\psi_l(\mathbf{e}^l)$ for each l.

Next, for each $\mathbf{a} \in \mathrm{atoms}(I)$ let $\lambda(\mathbf{a})$ be the set of those indices l such that $\mathbf{e}^l$ lies entirely in $M^\mathbf{a}$ and let $\mathbf{e}^{(\mathbf{a})}$ enumerate (without repetition) all elements from those $\mathbf{e}^l$ with $l \in \lambda(\mathbf{a})$. Let in addition $\delta^\mathbf{a}$ be the conjunction of all those formal equalities $a_j = (\mathbf{e}^{(\mathbf{a})})_k$ that hold in M. Further let $\nu^\mathbf{a} = \delta^\mathbf{a} \wedge \bigwedge_{l \in \lambda(\mathbf{a})} \alpha_l(\mathbf{e}^l) \wedge \neg\psi_l(\mathbf{e}^l)$ and $\psi^\mathbf{a} = \exists \mathbf{e}^{(\mathbf{a})} \nu^\mathbf{a}$.

Thus, $M \models \nu^\mathbf{a}(\mathbf{a}, \mathbf{e}^{(\mathbf{a})})$ and we find, as in (i), that also $M^\mathbf{a} \models \nu^\mathbf{a}(\mathbf{a}, \mathbf{e}^{(\mathbf{a})})$, hence $M^\mathbf{a} \models \psi^\mathbf{a}(\mathbf{a})$ and, because $M^\mathbf{a}, \mathbf{a} \equiv^r_w J, \mathbf{a}$, we also learn that $J \models \psi^\mathbf{a}(\mathbf{a})$. So there are $\mathbf{u}^{(\mathbf{a})}$ in J such that $J \models \nu^\mathbf{a}(\mathbf{a}, \mathbf{u}^{(\mathbf{a})})$.

Let $\mathbf{u}$ enumerate all $\mathbf{u}^{(\mathbf{a})}$ for $\mathbf{a} \in \mathrm{atoms}(I)$ different from $\mathbf{b}$. Note that, crucially, $|\mathbf{u}| \leq w$. Indeed, because for each $\mathbf{a}$ elements of $\mathbf{a}$ and $\mathbf{u}^{(\mathbf{a})}$ satisfy the equalities prescribed in $\delta^\mathbf{a}$, it is ensured that any equalities between elements of witnessing tuples $\mathbf{e}^{(\mathbf{a})}$ in $M^\mathbf{a}$ and $\mathbf{e}^{(\mathbf{a}')}$ in $M^{\mathbf{a}'}$ with $\mathbf{a} \neq \mathbf{a}'$ (which, by definition of M, must necessarily involve elements occurring both in $\mathbf{a}$ and in $\mathbf{a}'$) are also observed by the corresponding witnesses $\mathbf{u}^{(\mathbf{a})}$ and $\mathbf{u}^{(\mathbf{a}')}$ in J.

Altogether we have $J \models \bigwedge_{\mathbf{a} \neq \mathbf{b}} \nu^\mathbf{a}(\mathbf{a}, \mathbf{u}^{(\mathbf{a})})$, where $\mathbf{a}$ ranges over $\mathrm{atoms}(I) \setminus \{\mathbf{b}\}$. Let $\delta^\mathbf{b}$ be the conjunction of all equalities $b_j = u_k$ that do hold in J, and let $\zeta(\mathbf{b}, \mathbf{u}) = \delta^\mathbf{b} \wedge \bigwedge_{\mathbf{a} \neq \mathbf{b}} \bigwedge_{l \in \lambda(\mathbf{a})} \alpha_l(\mathbf{e}^l) \wedge \neg\psi_l(\mathbf{e}^l)$, where $\mathbf{a}$ ranges over $\mathrm{atoms}(I) \setminus \{\mathbf{b}\}$. By the above, $J \models \exists \mathbf{u}\, \zeta(\mathbf{b}, \mathbf{u})$ and

$\exists \mathbf{u}\, \zeta(\mathbf{b}, \mathbf{u}) \in \mathrm{DNF}^q_w$, so from $M^{\mathbf{b}}, \mathbf{b} \equiv^r_w J, \mathbf{b}$ it follows that $M^{\mathbf{b}} \models \exists \mathbf{u}\, \zeta(\mathbf{b}, \mathbf{u})$. Taking into account that δ stipulates all equalities between components of $\mathbf{b}$ and any witnesses $\mathbf{u}$ to ζ in $M^{\mathbf{b}}$ that are valid in J and, correspondingly, that are valid in M between elements of $\mathbf{b}$ and the original witnesses $\mathbf{c}$ to ψ in M, we can conclude from this and from of course $M^{\mathbf{b}} \models \psi^{\mathbf{b}}(\mathbf{d})$ that $M^{\mathbf{b}} \models \psi(\mathbf{d})$ as needed. This completes the induction step in the proof of Claim 1. □

A.7 Proof of Theorem 7.7

In the proof below, we concentrate on boolean queries. The PTime upper bound for boolean queries extends immediately to non-boolean queries: consider a k-ary SGNQ $\phi(x_1, \dots, x_k)$. Let $P_1, \dots, P_n$ be fresh monadic predicates. Then the problem of testing whether $I \models_{OWA} \phi(a_1, \dots, a_k)$, for given I and $a_1, \dots, a_k$, reduces to the problem whether $I' \models_{OWA} \exists x_1, \dots, x_k(\phi(x_1, \dots, x_k) \wedge P_1(x_1) \wedge \dots \wedge P_k(x_k))$, where I' extends I by interpreting each new predicate P_i by the singleton set $\{a_i\}$.

Proof. Consider a boolean SGNQ Q. By prescription, conjunctions under an even number of negations in Q may contain at most one conjunct that is a negated subformula. To allow for a uniform treatment we introduce a fresh nullary predicate `false` and add $\neg$`false` as a new conjunct in those subformulas of Q (whether in the context of an even or odd number of negations), where there were no negative conjuncts. After this trivial transformation Q takes the form

$$\texttt{false} \ \vee \ \bigvee_i \exists \mathbf{x}\, (\alpha_i(\mathbf{x}) \wedge \neg \exists \mathbf{y}\ \psi_i(\mathbf{x}, \mathbf{y})) \tag{4}$$

where each α_i is a conjunction of atoms and ψ_i is a DNF formula built with only $\exists$, $\wedge$ and guarded negation. We allow above $|\mathbf{x}| = 0$ and α_i to be an empty conjunction, i.e. vacuously true. Thus, (4) contains, as a special case, disjuncts of the form $\neg \exists \mathbf{y}\ \psi(\mathbf{y})$. As another special case, (4) may contain disjuncts $\exists \mathbf{x}\ (\alpha_i(\mathbf{x}) \wedge \neg \texttt{false})$ with $\psi_i = \texttt{false}$ and the corresponding quantification $\exists \mathbf{y}$ being vacuous ($|\mathbf{y}| = 0$). We shall write Q equivalently as

$$\bigwedge_i \forall \mathbf{x}\, (\alpha_i(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi_i(\mathbf{x}, \mathbf{y})) \ \rightarrow \ \texttt{false} \tag{5}$$

akin to a formulation of CQ entailment of frontier-guarded tgds – except for the fact that $\psi_i(\mathbf{x}, \mathbf{y})$ need not be quantifier free. Via induction on the quantifier alternation rank we show that (5) can be 'flattened' to an equi-satisfiable $\forall\exists$-formula of GNFO asserting that a conjunction of frontier-guarded tgds entails `false`.

Each ψ_i is in disjunctive normalform and occurs in the scope of an odd number of negations in the disjunction-free ϕ. Hence it assumes the following general form

$$\psi_i(\mathbf{x}, \mathbf{y}) = \gamma_i \ \wedge \ \bigwedge_m \neg \delta_{i,m} \ \wedge \ \bigwedge_n \neg \exists \mathbf{z}\ (\gamma'_{i,n} \wedge \neg \xi_{i,n})$$

where γ_i and $\gamma'_{i,n}$ are conjunctions of positive atoms, each $\delta_{i,m}$ is an atom and each $\xi_{i,m}$ is either an atom, or an existentially quantified formula, or `false`. Let $W_i(\mathbf{x}, \mathbf{y})$ be a new predicate symbol of the same arity as ψ_i. We replace in (5) the i-th conjunct with a collection of new conjuncts:

$$\begin{array}{ll}
\forall \mathbf{x}\ \ (\alpha_i(\mathbf{x}) \rightarrow \exists \mathbf{y}\ W_i(\mathbf{x}, \mathbf{y})) & \\
\forall \mathbf{xy}\ \ (W_i(\mathbf{x}, \mathbf{y}) \rightarrow \gamma_i) & \\
\forall \mathbf{xy}\ \ (W_i(\mathbf{x}, \mathbf{y}) \wedge \delta_{i,m} \rightarrow \texttt{false}) & \text{for each } \delta_{i,m} \\
\forall \mathbf{xyz}\ \ \big(W_i(\mathbf{x}, \mathbf{y}) \wedge \gamma'_{i,n} \rightarrow \xi_{i,n}\big) & \text{for each } \gamma'_{i,n}
\end{array}$$

Because all negations were properly guarded, all of these rules are frontier-guarded tgds, save perhaps some of those of the last kind with ξ an existentially quantified formula, which then pertain to the same restrictions as the original conjuncts of (5), only having a lower quantifier alternation rank. Iterating this transformation one eventually arrives at the desired form comprising only frontier-guarded tgds.

The theorem follows from the fact that OWA query answering against frontier-guarded tgds has PTime data complexity [5]. In fact, [4] shows that every CQ can be rewritten relative to a set of frontier-guarded tgds into a Datalog program that can be executed on a database instance to yield the OWA answer to the original query. Thus, each serial GNFO query Q can also be reformulated as a Datalog program $(\Pi, \texttt{false})$ such that $I \models_{\mathrm{OWA}} Q \iff \Pi(I) \models \texttt{false}$ for all instances I. □

A.8 Proof of Theorem 7.8

Proof. We combine ideas of [22] and [11, Theorem 15] to encode computations of a fixed Turing machine with the database instances representing input words. Using guarded tgds and an egd (in fact, a key constraint), we will force the existence of a grid frame onto which a valid computation of the Turing machine is charted. The advantage of using tgds and egds is the well-known fundamental principle [26, 19, 31] that open-world query answering on an instance D relative to a set Σ of tgds and egds reduces to query evaluation on the single universal (though generally infinite) chase model $chase(D, \Sigma)$. While the chase model with respect to guarded tgds is always tree-like [11, 5], the additional key constraint imposed by the egd enforces a grid-like structure of the chase model.

Let M be a Turing machine, to be chosen later, having tape alphabet A, states Q and transition function $\delta : Q \times A \rightarrow Q \times A \times \{-1, 0, 1\}$. Input words to M will be presented as successor-structures: comprising a *succ*-chain of A-labelled elements. The signature consists of a binary relation *succ* and unary relations P_a for every $a \in A$. We assume that the predicates P_a partition the input structure, that A contains a special start symbol $\triangleright$ labelling only the first element and a special blank symbol $\flat$ labelling only the last element of the successor-chain.

Next we define a set Σ_M of guarded tgds and egds over an expanded signature responsible for simulating M. Σ_M has as conjuncts the following guarded tgds (omitting the implicit universal quantification of variables in rule bodies).

$$\begin{array}{ll}
succ(x, y) \rightarrow \exists z\ succ(y, z) & succ(x, y) \wedge P_\flat(x) \rightarrow P_\flat(y) \\
succ(x, y) \rightarrow \exists uv\ cell(x, y, u, v) & \\
cell(x, y, u, v) \rightarrow next(x, u) \wedge next(y, v) \wedge succ(u, v) &
\end{array}$$

In addition, Σ_M contains the key constraint

$$next(x, y) \wedge next(x, z) \rightarrow y = z \tag{6}$$

expressing functionality of *next*. It is easy to see that the infinite chase of any input structure as specified above wrt. these guarded tgds and the egd is an infinite grid with *succ* and *next* acting as horizontal and vertical successor edges and whose bottom *succ*-chain is labelled with $\triangleright w \flat^\omega$, where w is the input word. Consequently, every model of these rules embeds a homomorphic image of this grid.

The next step is to implement, given the grid frame, the workings of the Turing machine M using additional guarded tgd rules. To this end the we will make use of additional unary predicates S_q for every state $q \in Q$ of M. Let *init* be the initial state and *acc* the w.l.o.g. unique accepting state of M. To initiate the computation Σ_M specifies

$$P_{\triangleright}(x) \to S_{init}(x)$$

and to carry it on Σ_M contains guarded tgds associated to each transition $(p, a, q, b, \iota) \in \delta$.

$$\begin{array}{ll} cell(x,y,u,v) \wedge S_p(x) \wedge P_a(x) \to P_b(x) \wedge S_q(u) & \text{for } \iota = 0 \\ cell(x,y,u,v) \wedge S_p(x) \wedge P_a(x) \to P_b(x) \wedge S_q(v) & \text{for } \iota = 1 \\ cell(x,y,u,v) \wedge S_p(y) \wedge P_a(y) \to P_b(y) \wedge S_q(u) & \text{for } \iota = -1 \end{array}$$

To ensure that tape symbols not affected by a transition are copied from one configuration to the next we add unary predicates L and R (intuitively, L and R mark the positions left and right of the head of a configuration, respectively) and the following guarded tgd rules.

$$\begin{array}{ll} succ(x,y) \wedge S_q(x) \to R(y) & succ(x,y) \wedge R(x) \to R(y) \\ succ(x,y) \wedge S_q(y) \to L(x) & succ(x,y) \wedge L(y) \to L(x) \end{array}$$
$$cell(x,y,u,v) \wedge R(y) \wedge P_a(y) \to P_a(v)$$
$$cell(x,y,u,v) \wedge L(x) \wedge P_a(x) \to P_a(u)$$

This completes the specification of the set Σ_M of guarded tgds and the single egd responsible for simulating the Turing machine M. It should be clear that M accepts a word w if, and only if, the corresponding instance D_w satisfies

$$D_w, \Sigma_M \models_{\text{OWA}} \exists x \; S_{acc}(x) \; .$$

The first claim of the theorem now follows by choice of some Turing machine M that accepts an r.e.-complete language. For the second claim consider the key constraint (6) and the GNFO query $\varphi_M \vee \exists x \; S_{acc}(x)$, where φ_M is the disjunction of the negations of the guarded tgds of Σ_M. □

A.9 Proof of Proposition 8.5

Proof. For the equivalence between (v) and (vi) we merely note that the stage increments $X^{n+1} \setminus X^n$ for each IDB predicate X are GNFO-definable, for each $n \in \mathbb{N}$. Now Π is classically unbounded if, and only if, for at least one X, these formulas are individually satisfiable, for every $n \in \mathbb{N}$; and similarly in restriction to finite instances. The finite model property for GNFO therefore shows the equivalence.

(ii) ⇒ (iv) and (i) ⇒ (iii) follow from the characterizations of GNFO as a fragment of FO in terms of preservation under suitable notions of guarded negation bisimulation (w-bounded guarded negation bisimulation), as presented in [7] for the classical version and in [33] for the finite model theory version. These apply since all stages of Π (finite and infinite, if we admit infinite instances) and especially the limit Π^∞ are preserved under w-bounded guarded negation bisimulation, if Π is of width w.

(iv) ⇒ (vi) is the natural variant of the classical Barwise–Moschovakis theorem for GNFO, which may be obtained from the classical via the semantic characterisation of GNFO as a fragment of FO in [7].

We concentrate on (iii) ⇒ (iv). Assume that formulas $\psi_X(\mathbf{x}_i) \in$ GNFO define X^∞ across all finite instances, but fail to define $\mathbf{X}^\infty = \Pi^\infty(I)$ over some infinite instance I. Appealing to the form of the rules in Π, we assume w.l.o.g. that $\psi_X(\mathbf{x})$ is explicitly guarded in the form $\psi_X(\mathbf{x}) = \bigvee_s \big(\alpha_s(\mathbf{x}_s) \wedge \psi_X(\rho_s(\mathbf{x}))\big)$, where every rule in Π with head predicate X gives rise to one disjunct, and ρ_s is the appropriate substitution to match the variable tuple $\mathbf{x}$ onto the $\mathbf{x}_s$ used in that rule (in particular, α_i guards all free variables in $\psi_X(\rho_s(\mathbf{x}))$).

The fact that a tuple of predicates $\mathbf{P}$ is a fixed point of Π is expressible by a sentence $\chi \in$ GNFO (in the signature extended with new P_X, one for each IDB predicate X in $\mathbf{X}$, which may even be used as guards). But for the tuple or predicates defined by the ψ_X, there is even a sentence $\xi \in$ GNFO in the basic (EDB) signature saying that this tuple is a fixed point of Π: the crucial point to note is that these predicate equalities reduce to set inclusions under each one of the relevant guards α_s (!). If one of the ψ_X failed over any infinite instance, then, by the finite model property for GNFO, it would also fail over some finite instance. So the ψ_X must define a fixed point of Π across all, finite and infinite, instances. A similar argument shows that this fixed point defined by the ψ_X over an infinite instance I must be the least fixed point $\mathbf{X}^\infty$. Otherwise, there would have to be some other, strictly smaller fixed point $\mathbf{P}$ (viz. $\mathbf{P} := \mathbf{X}^\infty$). This fact can also be expressed by a sentence of GNFO in the signature extended by the new predicate letters P. So the finite model property for GNFO would again pull this situation down to some finite instance – contradicting the assumption that the ψ_X define $\mathbf{X}^\infty$ over all finite instances. □

A.10 Proof of Theorem 9.1

Proof. It is known that the implication problem for inclusion dependencies and key constraints lacks finite controllability, and is undecidable both on finite and on unrestricted instances (cf. [1]). It follows that also the satisfiability and query containment problems for GN-SQL are not finitely controllable, and are undecidable both on finite instances and on unrestricted instances. As for the last item, it follows from Theorem 7.8, using the fact that key constraints (being a special case of functional dependencies) can be expressed in GN-SQL($\neq$). Specifically, the GN-SQL($\neq$) query for which open world query answering is undecidable, is q_1 union q_2 where q_1 is the boolean GN-SQL query from Theorem 7.8(ii) and q_2 is the boolean GN-SQL($\neq$) query expressing the negation of the key constraint from Theorem 7.8(ii). □

A.11 Proof of Theorem 9.2

Proof (sketch). In translating GN-SQL(lin) to GNFO, we have to overcome a discrepancy in the use of constants. The constants that may appear in a GN-SQL(lin) query are actual values from the linearly ordered domain lin. GNFO, on the other hand, allows for the use of constant symbols, whose interpretation is given by the structure, and may differ between structures. In particular, a structure may interpret two constant symbols by the same element. In order to overcome this discrepancy, we (i) introduce for each element d of lin a corresponding constant symbol d, and (ii) we construct a GNFO sentence that "axiomatizes" the correct behavior of the constant symbols (including the fact that distinct constant symbols denote different values).

More precisely, let lin $= (D, \prec)$ and for each finite subset $S = \{d_1, \ldots, d_n\}$ of D, with $d_1 \prec \ldots \prec d_n$, let θ_S be the following GNFO sentence, containing a constant symbol d_i for each $d_i \in S$:

$$\forall x \Big(\phi_{x<\mathsf{d}_1} \vee \phi_{x=\mathsf{d}_1} \vee \phi_{\mathsf{d}_1<x<\mathsf{d}_2} \vee \phi_{x=\mathsf{d}_2} \vee \cdots \vee \phi_{x>\mathsf{d}_n} \Big)$$

where

$$\phi_{x<\mathsf{d}_1} = \begin{cases} \bigwedge_{i\leq n}(x < \mathsf{d}_i \wedge \neg(x = \mathsf{d}_i) \wedge \neg(\mathsf{d}_i < x)) & \text{if } \exists d \in D\ d \prec d_1 \\ \bot & \text{otherwise} \end{cases}$$

$$\begin{aligned}\phi_{x=\mathsf{d}_i} = {} & (x = \mathsf{d}_i) \wedge \neg(x < \mathsf{d}_i) \wedge \neg(\mathsf{d}_i < x) \\ & \wedge \bigwedge_{j<i}(\mathsf{d}_j < x \wedge \neg(\mathsf{d}_j = x) \wedge \neg(x < \mathsf{d}_j)) \\ & \wedge \bigwedge_{j>i}(x < \mathsf{d}_j \wedge \neg(\mathsf{d}_j = x) \wedge \neg(\mathsf{d}_j < x))\end{aligned}$$

$$\phi_{\mathsf{d}_i<x<\mathsf{d}_{i+1}} = \begin{cases} \bigwedge_{j\leq i}(\mathsf{d}_j < x \wedge \neg(\mathsf{d}_j = x) \wedge \neg(x < \mathsf{d}_j)) \\ \quad \wedge \bigwedge_{j>i}(x < \mathsf{d}_j \wedge \neg(\mathsf{d}_i = x) \wedge \neg(\mathsf{d}_j < x))) \\ \qquad\qquad\qquad \text{if } \exists d \in D\ d_i \prec d \prec d_{i+1} \\ \bot \qquad\qquad\qquad \text{otherwise} \end{cases}$$

$$\phi_{\mathsf{d}_n<x} = \begin{cases} \bigwedge_{i\leq n}(\mathsf{d}_i < x \wedge \neg(x = \mathsf{d}_i) \wedge \neg(x < \mathsf{d}_i)) & \text{if } \exists d \in D\ d_n \prec d \\ \bot & \text{otherwise} \end{cases}$$

Observe that this set θ_S can be constructed from S in polynomial time, since LIN is reasonable. Furthermore, the following crucial property holds: if M is any structure satisfying θ_S, and if M' is an isomorphic copy of M in which each constant symbol d_i denotes the actual corresponding value d_i (for all $d_i \in S$), then M and M' are indistinguishable with respect to GN-SQL(LIN) queries whose constants are included in S. It follows that a GN-SQL(LIN) query q_1 is contained in a GN-SQL(LIN) query q_2 if and only if, for their GNFO translations q_1^* and q_2^*, we have that $q_1^* \wedge \theta_S \models q_2^*$, where S is the set of constants occurring in q_1 and q_2. □

A.12 Proof of Proposition 9.3

PROOF. The upper bound follows from the ExpTime upper bound for stratified Datalog [18] (obtained by the standard technique of "grounding" a program by instantiating its rules via substituting domain elements for the variables in every possible way and solving the resulting exponentially large propositional Horn program with stratified negation by standard means).

For the lower bound we provide a reduction from the acceptance problem for polynomial-space alternating Turing machines. Consider an alternating Turing machine M using $p(n)$ tape cells on any input of length n. We may assume w.l.o.g. that the states of M are partitioned into existential states $Q_\exists$ and universal states $Q_\forall$ and that the transition table of M consists of tuples $(p, a, q, b, \epsilon, s, c, \delta)$ interpreted as follows. When in state $p \in Q$ and reading a there are two possible transitions: writing b at the current position, entering state q, and moving the read-write head by $\epsilon \in \{-1, 0, +1\}$; or writing c, entering state s and moving the head by $\delta \in \{-1, 0, +1\}$. The choice between the two possibilities is existential or universal according to whether $p \in Q_\exists$ or $p \in Q_\forall$. In addition we may assume w.l.o.g. that M has a unique initial state *init*, and a unique accepting configuration with state *acc*, head position 1 and the used segment of its tape filled with 0's.

Let $\mathfrak{B}$ be the structure with domain $\{0, 1\}$ and the binary relation *Bits* that holds the pair $(0, 1)$ alone. Given an input word w of length $|w| = n$ and M as above we let $N = p(n)$ and we devise a GN-Datalog program with IDB predicates $S_{q,i}(u_1, \ldots, u_N, z, o)$ of arity $N + 2$ for each $q \in Q_\exists \cup Q_\forall$ and $1 \leq i \leq N$. Intuitively speaking, every fact $S_{q,i}(u_1, \ldots, u_N, z, o)$ will encode a configuration of M in state q, head location i and tape contents $u_1 \ldots u_N$, and z and o will invariably contain the values 0 and 1 in every such fact ever derived. The GN-Datalog program $\Pi_{M,n}$ simulating M on an $N = p(n)$-bounded tape comprises the following rules. For every transition $(p, a_0, q, a_1, \epsilon, s, a_2, \delta)$ and every i such that $1 \leq i, i + \epsilon, i + \delta \leq N$, if $p \in Q_\exists$ then there are rules

$$\begin{aligned} & S_{p,i}(u_1, \ldots, u_{i-1}, \sigma_0, u_{i+1}, \ldots, u_N, z, o) \\ & \quad \leftarrow\ S_{q,i+\epsilon}(u_1, \ldots, u_{i-1}, \sigma_1, u_{i+1}, \ldots, u_N, z, o)\,. \\ & S_{p,i}(u_1, \ldots, u_{i-1}, \sigma_0, u_{i+1}, \ldots, u_N, z, o) \\ & \quad \leftarrow\ S_{s,i+\delta}(u_1, \ldots, u_{i-1}, \sigma_2, u_{i+1}, \ldots, u_N, z, o)\,. \end{aligned}$$

and if $p \in Q_\forall$ then there is a rule

$$\begin{aligned} & S_{p,i}(u_1, \ldots, u_{i-1}, \sigma_0, u_{i+1}, \ldots, u_N, z, o) \\ & \quad \leftarrow\ S_{q,i+\epsilon}(u_1, \ldots, u_{i-1}, \sigma_1, u_{i+1}, \ldots, u_N, z, o), \\ & \qquad S_{s,i+\delta}(u_1, \ldots, u_{i-1}, \sigma_2, u_{i+1}, \ldots, u_N, z, o)\,. \end{aligned}$$

where in both cases σ_j is $\begin{cases} z & \text{if } a_j = 0 \\ o & \text{if } a_j = 1 \end{cases}$ for each $j = 0, 1, 2$. In addition there is an acceptance rule

$$S_{acc,1}(\underbrace{z, \ldots, z}_{N \text{ times}}, z, o) \quad \leftarrow \quad Bits(z, o)$$

corresponding to the unique accepting configuration, and the answer rule

$$Ans_w \quad \leftarrow \quad S_{init,1}(u_1, \ldots, u_N, z, o), Bits(z, o)$$

encoding the initial configuration for a given input word $w \in \{0, 1\}^n$, where each u_i is one of the variables z or o according to whether the ith bit of the initial tape contents with input w is zero or one. Then w is accepted by M if the GN-Datalog query $(\Pi_{M,|w|}, Ans_w)$ evaluates to true on $\mathfrak{B}$. □