Rewriting Queries Using Views with Access Patterns Under Integrity Constraints

Alin Deutsch

Department of Computer Science and Engineering,
University of California San Diego
9500 Gilman Drive, La Jolla, CA 92093, U.S.A.
deutsch@cs.ucsd.edu

Bertram Ludäscher

Department of Computer Science
University of California Davis
One Shields Avenue, Davis, CA 95616, U.S.A.
ludaesch@ucdavis.edu

Alan Nash

Department of Computer Science and Engineering,
University of California San Diego
9500 Gilman Drive, La Jolla, CA 92093, U.S.A.
anash@san.rr.com

Abstract

We study the problem of rewriting queries using views in the presence of access patterns, integrity constraints, disjunction, and negation. We provide asymptotically optimal algorithms for (1) finding minimally containing and (2) maximally contained rewritings respecting the access patterns (which we call *executable*) and for (3) deciding whether an exact executable rewriting exists. We show that rewriting queries using views in this case reduces (a) to rewriting queries with access patterns and constraints without views and also (b) to rewriting queries using views under constraints without access patterns. We show how to solve (a) directly and how to reduce (b) to rewriting queries under constraints only (semantic optimization). These reductions provide two separate routes to a unified solution for problems 1, 2, and 3 based on an extension of the relational chase theory to queries and constraints with disjunction and negation. We also handle equality and arithmetic comparisons. We also show that in an information integration setting, maximally contained rewritings are given by the certain answers (under the usual semantics) for a set of constraints derived from the binding patterns. That is, except for defining

the appropriate constraints, binding patterns do not need special treatment. Finally, we show that if there is an exact executable rewriting, there is an executable rewriting which is a union of conjunctive queries with negation.

1 Introduction

We study the problem of rewriting a query Q in terms of a given set of views $\mathcal{V}$ with limited access patterns $\mathcal{P}$, under a set Σ of integrity constraints. More precisely, we are interested in determining whether there exists a query plan Q', expressed in terms of the views $\mathcal{V}$ only, that is executable (i.e., observes $\mathcal{P}$) and equivalent to Q for all databases satisfying Σ. If there is no such Q', then we seek the minimally containing and maximally contained executable queries, which provide the "best possible" executable query plans for approximating the answer to Q from above and below. Our results unify and extend a number of previous results in data integration (see related work). In particular, they apply to queries, views, and constraints over unions of conjunctive queries with negation ($\text{UCQ}^\neg$), equality and arithmetic comparisons.

The following example shows the common case of a query that has no equivalent executable rewriting (i.e., is not feasible) in the absence of constraints, but that can yield such a rewriting when constraints are given.

Example 1 Consider the following set of relations with access patterns: conference $C^{\text{io}}(a,t)$, journal $J^{\text{io}}(a,t)$, magazine $M^{\text{oo}}(a,t)$, PC-magazine $P^{\text{ioo}}(a,t,p)$, the set of listed publishers $L^{\text{i}}(p)$, repository $R^{\text{oo}}(a,t)$, ACM anthology $A^{\text{iii}}(a,t,o)$, and DBLP conference article $D^{\text{ooo}}(a,t,c)$. The relation symbols are annotated with access patterns, indicating which arguments must be given as inputs (marked 'i') and which ones can be retrieved as outputs (marked 'o') when accessing the relation. For example $C^{\text{io}}(a,t)$ means that an author a has to be given as input before one can retrieve the titles t of a's conference publications from $C(a,t)$.

Let Q be the query which asks for pairs of authors and titles of conference publications, journal publications, and magazines which are not PC-magazines:

$$Q(a,t) :- \underline{C(a,t)} \tag{1}$$
$$Q(a,t) :- \underline{J(a,t)} \tag{2}$$
$$Q(a,t) :- M(a,t), \underline{\neg P(a,t,p)}, \underline{L(p)} \tag{3}$$

[1] A preliminary version of this paper appeared in [3].

(We restrict the publishers to those in L to make the query safe.) Q cannot be executed since no underlined literal is answerable: e.g., the access patterns require a to be bound before invoking $C(a,t)$ but no such binding is available. Worse yet, Q is not even feasible, i.e., there is no executable query Q' equivalent to Q. However, if the following set Σ of integrity constraints is given, an executable Q' can be found that is equivalent under Σ:

$$\forall a \forall t\ C(a,t) \rightarrow \exists c\ D(a,t,c) \tag{4}$$

$$\forall a \forall t\ J(a,t) \rightarrow \exists p\ R(a,t) \wedge \neg P(a,t,p) \wedge L(p) \vee\ \exists o \exists c\ A(a,t,o) \wedge D(a,t,c) \tag{5}$$

$$\forall a \forall t\ M(a,t) \rightarrow \exists p\ \neg P(a,t,p) \wedge L(p) \tag{6}$$

Constraint (4) states that every conference publication is a DBLP conference publication; (5) states that every journal publication is available from a repository, is not a PC magazine, but comes from a listed publisher, or is available from the ACM anthology and from DBLP; and (6) states that magazine articles are not PC-magazine articles. We are only interested in databases which satisfy these constraints Σ. On those databases, Q is equivalent to Q^Σ, obtained by "chasing" Q with Σ:

$$Q^\Sigma(a,t) :- C(a,t), D(a,t,c)$$
$$Q^\Sigma(a,t) :- J(a,t), R(a,t), \underline{\neg P(a,t,p)}, \underline{L(p)}$$
$$Q^\Sigma(a,t) :- J(a,t), \underline{A(a,t,o)}, D(a,t,c)$$
$$Q^\Sigma(a,t) :- M(a,t), \underline{\neg P(a,t,p)}, \underline{L(p)}$$

Again, unanswerable literals are underlined. The *answerable part* $\mathrm{ans}(Q^\Sigma)$ is obtained (roughly) by removing unanswerable parts (see Definition 8 for details):

$$\mathrm{ans}(Q^\Sigma)(a,t) :- D^{\mathrm{ooo}}(a,t,c), C^{\mathrm{io}}(a,t) \tag{7}$$

$$\mathrm{ans}(Q^\Sigma)(a,t) :- R^{\mathrm{oo}}(a,t), J^{\mathrm{io}}(a,t) \tag{8}$$

$$\mathrm{ans}(Q^\Sigma)(a,t) :- D^{\mathrm{ooo}}(a,t,c), J^{\mathrm{io}}(a,t) \tag{9}$$

$$\mathrm{ans}(Q^\Sigma)(a,t) :- M^{\mathrm{oo}}(a,t) \tag{10}$$

In general, the answerable part is not equivalent to Q: e.g., the subquery (10) is not contained in (3) and thus $\mathrm{ans}(Q^\Sigma)$ might produce more answers than Q. However the equivalence may still hold under Σ, i.e., for all databases satisfying Σ. This can be checked (cf. Corollary 4) and is indeed the case here. Then $\mathrm{ans}(Q^\Sigma)$ is the desired executable plan, equivalent to Q for all databases satisfying the constraints Σ. $\square$

As we will show, if there is an equivalent query Q' under Σ, our algorithm

will find it, and if no such Q' exists, we can find the minimally containing and the maximally contained plans, providing least overestimate and greatest underestimate queries for Q under Σ, respectively.

Example 2 This example illustrates that our techniques can also rewrite queries in terms of views with access patterns. For example, the rules

$$V_1^{\text{oo}}(a,t) :- C(a,t), R(a,t)$$
$$V_2^{\text{io}}(a,t) :- C(a,t), \neg R(a,t)$$

state that the view V_1 has conference articles that are also in the repository R, while V_2 has those that are not in R. The access patterns indicate that at least a must be given when accessing $V_2(a,t)$, while no inputs are required for accessing V_1. We will show that if we want to rewrite a query in terms of the views only, this can be achieved by considering constraints and access patterns only. To this end, we model views as constraints and also include "negation constraints" of the form $\forall a \forall t\,(\texttt{true} \rightarrow (R(a,t) \vee \neg R(a,t)))$. Chasing the query $Q(a,t) :- C(a,t)$ with the latter yields

$$Q'(a,t) :- C(a,t), R(a,t)$$
$$Q'(a,t) :- C(a,t), \neg R(a,t)$$

which then rewrites in terms of V_1 and V_2 to

$$Q''(a,t) :- V_1^{\text{oo}}(a,t)$$
$$Q''(a,t) :- V_2^{\text{io}}(a,t).$$

Here, Q'' is not executable (the access pattern for V_2 requires a to be bound). Under the constraint $\forall a \forall t\,(C(a,t) \rightarrow R(a,t))$, our algorithm can discard the unanswerable second rule, resulting in the executable rewriting $Q'''(a,t) :- V_1(a,t)$. □

Contributions. We solve the problem of rewriting queries using views with limited access patterns under integrity constraints (denoted $\{Q, \mathcal{V}, \mathcal{P}, \Sigma\}$) and prove that feasibility is **NP**-complete in the size of the input queries, for fixed views and query inclusion constraints over [2] UCQ and $\mathbf{\Pi_2^P}$-complete for UCQ$^\neg$. These results hold in those cases when the chase terminates and its result is not too large (Theorem 14). While checking for this is undecidable, A fairly general sufficient condition is given by Theorem 10. We present an algorithm, ViewRewrite, which is guaranteed to find an exact plan (if one exists) or

[2] A query inclusion constraint *over* $\mathcal{L}$ is an implication $\forall \bar{x}(U \rightarrow V)$ with $U, V \in \mathcal{L}$ (cf. Section 2).

at least the minimally containing plan (unique if it exists) (Theorem 13). We also give an algorithm for finding the maximally contained executable plan (Theorem 15). Moreover, we expose an interesting connection between computing the maximally contained executable plan and computing the certain answers in an information integration system (Theorem 18). We are particularly interested in complexity results given in terms of the size of the input query only, for fixed schema, views and constraints, because a typical system would be configured off-line for a given schema, views, and constraints, then process a large number of input queries.

One side effect of our results is a unified treatment for three flavors of rewriting problems which have been introduced and solved separately in prior work. We show that $\{Q, \mathcal{V}, \mathcal{P}, \Sigma\}$ reduces to $\{Q, \mathcal{P}, \Sigma'\}$, i.e., rewriting queries with access patterns and constraints without views (Theorem 13) and also to $\{Q, \mathcal{V}, \Sigma''\}$, i.e. rewriting queries under constraints using views without access patterns (Theorem 20).

We show how to solve $\{Q, \mathcal{P}, \Sigma'\}$ and $\{Q, \mathcal{V}, \Sigma''\}$ by reduction to rewriting queries under constraints only. These reductions provide two separate routes

$\{Q, \mathcal{V}, \mathcal{P}, \Sigma\} \rightsquigarrow \{Q, \mathcal{P}, \Sigma'\} \rightsquigarrow \{Q, \Sigma'''\}$ and
$\{Q, \mathcal{V}, \mathcal{P}, \Sigma\} \rightsquigarrow \{Q, \mathcal{V}, \Sigma''\} \rightsquigarrow \{Q, \Sigma''''\}$

to a unified solution for all three problems, based on our extension of the relational chase theory to queries and constraints with disjunction and negation (Section 3). Specifically we show that a minimally containing query in the $\{Q, \mathcal{P}, \Sigma\}$ case can be obtained by chasing Q with Σ and computing the answerable part. Similarly, in the presence of views, we can compute the minimally containing query by chasing with Σ and the constraints corresponding to $\mathcal{V}$ and again computing the answerable part.

We also extend the above results to handle equality and arithmetic comparisons by modeling them with constraints (Section 8).

Finally, since the notion of feasibility depends on the language in which the rewriting is to be expressed, it is natural to ask whether there are cases when there exists no rewriting of Q in the prescribed language, but Q is nevertheless answerable by a general computable query. We show the surprising result that this is not the case, that is, answerability and feasibility (for appropriately chosen rewriting language) coincide (Theorem 21).

Related Work. There is a large body of related work that deals with one or more of the following three aspects: (i) query rewriting under *limited access patterns*, see [23,21,15,9,24,17,16,20,19] and references within; (ii) query rewriting under *integrity constraints* (a.k.a. semantic query optimization), see

for instance [13,5] and references within; and (iii) query rewriting and answering *using views* [6,7,11]. These all have important applications in data integration and query optimization [14,18,10]. All of the above mentioned work on rewriting has focused on either of two flavors: *maximally contained* or *exact* rewritings.

In this paper, we introduce algorithms which deal uniformly with all three aspects of rewriting and which find exact, maximally contained and *minimally containing* rewritings.

In the category of maximally contained rewritings, the closest related results are those of [7], which considers the most expressive queries and views, and of [13], which handles the most expressive constraints. [7] shows how to obtain a maximally contained rewriting for recursive Datalog queries using conjunctive query views. [7] also considers access patterns on the views as well as very restricted constraints (which can express the standard key but not all foreign key constraints) and it shows how to construct a recursive plan which is guaranteed to be maximally contained. As opposed to [7], we do not consider recursive queries but we allow negation and disjunction in queries, views and constraints (our constraints express key, foreign key, join, multi-valued, and embedded dependencies and beyond). Moreover, we provide decision procedures for the existence of an *exact* plan and, in its absence, we show how to obtain not only the best *contained* but also the best *containing* approximations. [13] finds the maximally contained rewriting of CQ queries under more expressive constraints than [7] (embedded dependencies), provided the predicate dependency graph is acyclic. However, views, access patterns and negation (in either query or constraints) are not handled.

With respect to finding exact rewritings, [5] shows how to treat views and integrity constraints uniformly for UCQ queries. The present paper extends these results to $\text{UCQ}^\neg$ queries, constraints, views with limited access patterns, and maximally contained and minimally containing rewritings. [21,17,16] shows **NP**-completeness in the size of the query for deciding feasibility of UCQ queries over relations with limited access patterns (i.e. no negation, no views and no constraints are considered). Still in the absence of views and constraints, [20] shows that if negation is added then deciding feasibility becomes $\mathbf{\Pi_2^P}$-complete; [19] further extends the notion of feasibility to all first-order queries and characterizes the complexity of many first-order query classes.

Paper Outline. The preliminaries in Section 2 include earlier results on containment and feasibility under access patterns. Section 3 presents our extension of the chase procedure to unions of conjunctive queries with negation. The extended chase tool will be employed in the remainder of the paper for query rewriting. In particular, Section 4 presents our results on feasibility and

rewriting with access patterns under constraints. In Section 5 we generalize these results to include views. In Section 6 we establish our results on maximally contained executable queries. Section 7 provides an alternative method for deciding feasibility: Instead of handling access patterns via the answerable part of a query, we show that they too can be reduced to constraints and the chase. Section 8 shows how other extensions such as equality and arithmetic comparisons all can be treated uniformly via constraints. Finally, Section 9 defines a notion of answerability independent of a query language and shows that for the case of $\text{UCQ}^\neg$ queries, feasible is the same as answerable.

2 Preliminaries

2.1 Basics

A schema τ is a list of relation symbols and their arities. An instance A *over* σ has one relation for every relation symbol in σ, of the same arity. The universe of A, which we also denote A consists of all the values in all the relations of A. We write $|A|$ for the size of the universe of A. For an instance A and a relation symbol $R \in \tau$, we write R^A for the relation in A associated to R.

The complexity results in this paper assume the schema τ to be fixed, and that all queries, views and constraints are over τ.

2.2 Queries

A *term* is a variable or constant. We write $\bar{x}$ to denote a finite sequence of terms $x_1, \ldots, x_k$. We use lowercase letters $x, y, z, \ldots$ for terms and uppercase letters $P, Q, R, \ldots$ for relation symbols and queries. A *datalog rule* is an expression of the form $P(\bar{z}) :- \ell_1(\bar{x}_1), \ldots, \ell_n(\bar{x}_n)$ where each $\ell_i(\bar{x}_i)$ in the rule is a *literal*, i.e., a *positive* atom $R(\bar{x})$ or a *negative* literal $\neg R(\bar{x})$. Given a rule Q, we define $\text{head}(Q)$ and $\text{body}(Q)$ to be the parts to the left and to the right of $''{:-}''$, respectively. A datalog *program* is a finite set of datalog rules. We only consider *nonrecursive* programs and we further require that all rules have the same head. In particular, this implies that we do not allow atoms in the body of a rule which refer to the head of another rule. Therefore, $\text{head}(P)$ is well-defined for the programs P we consider.

We represent queries (and therefore views) by programs unless otherwise specified. If a query Q is given by multiple rules $Q_1, \ldots, Q_n$, we denote this by $Q = \bigvee_i Q_i$ and we have $Q(D) = \bigcup_i Q_i(D)$, where $Q(D)$ denotes the result of

query Q on database D.

Queries given by one or more rules are *unions of conjunctive queries with negation* ($\mathrm{UCQ}^\neg$). Those given by a single rule are *conjunctive queries with negation* ($\mathrm{CQ}^\neg$). If all literals are positive, then they are *unions of conjunctive queries* (UCQ) in the former case, and *conjunctive queries* (CQ) in the later case.

We say that a variable *appears positively* if it appears in a positive literal. A query $Q \in \mathrm{CQ}^\neg$ is *safe* if every variable which appears in the rule (whether in the head or in the body) appears positively in its body. A query $Q = \bigvee_i Q_i$ with $Q_1, \ldots, Q_n \in \mathrm{CQ}^\neg$ is *safe* if every Q_i is safe and all Q_is have the same head.

In the definition of $\mathrm{ans}(Q)$ below, we will need to consider two special kinds of queries. A query $Q \in \mathrm{CQ}^\neg$ given by $\mathrm{head}(Q) :- \texttt{false}$ is unsatisfiable and is always safe (this is an extension of the definition above). A query $Q \in \mathrm{CQ}^\neg$ given by a rule with an empty body is safe if there are no variables in the head (i.e., if the query is boolean).

Unless otherwise specified, all queries are assumed to be in $\mathrm{UCQ}^\neg$ and safe. Furthermore, E, P, and Q always denote queries.

2.3 Query Containment

This section contains a review of well-known results (see, in particular, [22] for the handling of negation) which we need in order to prove the results in the following sections. Its main purpose is to fix the notation and summarize the known results in this notation. We include some of the proofs since they help in the understanding of the sections that follow.

P is *contained* in Q ($P \sqsubseteq Q$) if, for all databases D, $P(D) \subseteq Q(D)$. P is *equivalent* to Q ($P \equiv Q$) if $P \sqsubseteq Q$ and $Q \sqsubseteq P$. Given a set of constraints Σ, P is Σ-*contained* in Q ($P \sqsubseteq_\Sigma Q$) if, for all D which satisfy Σ, $P(D) \subseteq Q(D)$. P is Σ-*equivalent* to Q ($P \equiv_\Sigma Q$) if $P \sqsubseteq_\Sigma Q$ and $Q \sqsubseteq_\Sigma P$.

P is *strictly contained* in Q ($P \sqsubset Q$) if $P \sqsubseteq Q$ and $Q \not\sqsubseteq P$. P *minimally contains* Q if $Q \sqsubseteq P$ and there is no P' such that $Q \sqsubset P' \sqsubset P$. P is *maximally contained* in Q if $P \sqsubseteq Q$ and there is no P' such that $P \sqsubset P' \sqsubset Q$. We define *minimally* Σ-*contains* and *maximally* Σ-*contained* similarly.

$\mathrm{CONT}(\mathcal{L})$ is the decision problem: for queries $P, Q \in \mathcal{L}$ determine whether $P \sqsubseteq Q$ ($\mathcal{L}$ is a class of queries). $\mathrm{CONT}_\Sigma(\mathcal{L})$ is the problem: for a fixed set of constraints Σ and queries $P, Q \in \mathcal{L}$ decide whether $P \sqsubseteq_\Sigma Q$.

Theorem 1

(1) CONT(CQ) *and* CONT(UCQ) *are* **NP**-*complete [2].*
(2) CONT($\text{CQ}^\neg$) *and* CONT($\text{UCQ}^\neg$) *are* $\mathbf{\Pi_2^P}$-*complete [22].*

We write free(Q) for the set of free variables in the query Q and vars(Q) for the set of all variables in the query Q (both free and existentially quantified).

Definition 1 [Homomorphism] Given $P, Q \in \text{CQ}^\neg$ with heads $P(\bar{x})$ and $Q(\bar{y})$, a mapping

$$h : \text{vars}(P) \cup \text{consts}(P) \rightarrow \text{vars}(Q) \cup \text{consts}(Q)$$

(which we write $h : P \rightarrow Q$ for brevity) is a *homomorphism* from P into Q if $h(\bar{x}) = \bar{y}$ and, for every literal $\ell(\bar{v})$ in P, there is a literal $\ell(h(\bar{v}))$ in Q. We write $P \hookrightarrow Q$ if there is a homomorphism from P to Q. Notice that we require homomorphisms to preserve *literals*, i.e. both positive and negated atoms.

Definition 2 [Satisfies] Given $Q \in \text{CQ}^\neg$ with free variables $\bar{x} := \text{free}(Q)$, a database D, and a tuple of constants $\bar{d}$, $D\bar{d}$ *satisfies* Q, which we write $D\bar{d} \models Q$ if there is a mapping

$$h : \text{vars}(Q) \cup \text{consts}(Q) \rightarrow D$$

such that h is the identity on the constants, $h(\bar{x}) = \bar{d}$, for every atom $R(\bar{v})$ in Q, the tuple $h(\bar{v})$ appears in relation R in D and, for every negated atom $\neg R(\bar{v})$ in Q, the tuple $h(\bar{v})$ does not appear in relation R in D

Definition 3 [Frozen Instance] If $Q \in \text{CQ}^\neg$, then $[Q] := D\bar{d}$, the *frozen instance* of Q, consists of a database D and a distinguished tuple $\bar{d}$. D is the database which consists of one tuple for each positive atom in Q where each variable x has been replaced by a corresponding constant c_x. $\bar{d}$ is the tuple which consists of the constants $c_{x_1} \ldots c_{x_k}$ corresponding to the free variables $x_1 \ldots x_k$ in Q.

We write $D\bar{d} \models Q$ instead of the more traditional $D \models Q[\bar{d}]$ because we often need to refer to frozen instances and these consist of both a database and a distinguished tuple of constants and therefore it is more convenient to write $[P] \models Q$ rather than something that would require to separate the database in $[P]$ from the distinguished tuple of constants in $[P]$. We set $Q(D) := \{\bar{d} : D\bar{d} \models Q\}$.

We say that $Q \in \text{CQ}^\neg$ is *satisfiable*, if there is $D\bar{d}$ such that $D\bar{d} \models Q$. Notice that a $\text{CQ}^\neg$ query Q is satisfiable iff there is no atom $R(\bar{x})$ which appears both positively and negatively and that checking for the presence of such atom can be done in quadratic time. The following two results follow directly from the corresponding definitions.

Lemma 1

(1) If $Q \in \mathrm{CQ}$, then $[Q] \models Q$.
(2) If $Q \in \mathrm{CQ}^{\neg}$ and Q is satisfiable, then $[Q] \models Q$.

Lemma 2

(1) If $P, Q, R \in \mathrm{CQ}^{\neg}$ and $P \hookrightarrow Q \hookrightarrow R$, then $P \hookrightarrow R$.
(2) If $P, Q \in \mathrm{CQ}^{\neg}$, $P \hookrightarrow Q$, and $D\bar{d} \models Q$, then $D\bar{d} \models P$.

Proof. (1) If $f : P \hookrightarrow Q$ and $g : Q \hookrightarrow R$, then $h : g \circ f$ is a homomorphism $P \hookrightarrow R$. (2) If there is a mapping $g : Q \to D$ as in Definition 2 and $f : P \hookrightarrow Q$, then $h : g \circ f$ is a mapping $P \to D$ satisfying Definition 2 so $D\bar{d} \models P$. □

Definition 4 [Complete Query] $P \in \mathrm{CQ}^{\neg}$ is *complete* if it is satisfiable and, for all $Q \in \mathrm{CQ}^{\neg}$, $[P] \models Q$ implies $Q \hookrightarrow P$. $P \in \mathrm{UCQ}^{\neg}$ is *complete* if $P = \bigvee_i P_i$ where each P_i is $\mathrm{CQ}^{\neg}$ and complete.

Intuitively, $P \in \mathrm{CQ}^{\neg}$ is complete if tuples not in $[P]$ (constructed from $[P]$'s active domain) correspond to negated atoms in P.

Example 3 Consider the queries

$$P(x, y) :- R(x, y), R(y, y)$$

and

$$Q(x, y) :- R(x, y), \neg R(y, x)$$

We have $[P] \models Q$, yet $Q \not\hookrightarrow P$ since there are no negative literals in P. This shows that P is not complete. On the other hand, the queries

$$P'(x, y) :- R(x, y), R(y, y), \neg R(y, x), R(x, x)$$

and

$$P''(x, y) :- R(x, y), R(y, y), R(y, x), R(x, x)$$

are both complete. We have $[P'] \models Q$ and $Q \hookrightarrow P'$ and we also have $[P''] \not\models Q$ and $Q \not\hookrightarrow P''$.

Lemma 3

(1) If $P, Q \in \mathrm{CQ}$, then $Q \hookrightarrow P$ iff $[P] \models Q$.
(2) If $P, Q \in \mathrm{CQ}^{\neg}$ and P is complete, then $Q \hookrightarrow P$ iff $[P] \models Q$.

Proof. (1) Assume $P, Q \in \mathrm{CQ}$ and $Q \hookrightarrow P$. Then $[P] \models P$ by Lemma 1 and therefore $[P] \models Q$ by Lemma 2. Conversely, assume $P, Q \in \mathrm{CQ}$ and $[P] \models Q$. Then the function h witnessing the latter (Definition 2) is a homomorphism $Q \hookrightarrow P$.

(2) Assume $P, Q \in \mathrm{CQ}^{\neg}$ and $Q \hookrightarrow P$. Then, since P is complete and therefore satisfiable, $[P] \models P$ by Lemma 1 and therefore $[P] \models Q$ by Lemma 2. Conversely, assume $P, Q \in \mathrm{CQ}^{\neg}$, P is complete, and $[P] \models Q$. Then $Q \hookrightarrow P$ follows directly from the definition of complete. $\square$

Theorem 2

(1) If $P, Q \in \mathrm{CQ}$, then $P \sqsubseteq Q$ iff $Q \hookrightarrow P$.
(2) If $P, Q \in \mathrm{CQ}^{\neg}$ and P is complete, then $P \sqsubseteq Q$ iff $Q \hookrightarrow P$.

Proof. (1) Assume $P, Q \in \mathrm{CQ}$. If $Q \hookrightarrow P$ and $D\bar{d} \models P$ then $D\bar{d} \models Q$ by Lemma 2. Therefore $P \sqsubseteq Q$. Conversely, since $[P] \models P$ always holds, if $P \sqsubseteq Q$, then $[P] \models Q$ and therefore $Q \hookrightarrow P$ by Lemma 3. (2) If $P, Q \in \mathrm{CQ}^{\neg}$, then the same proof works, except that we use the fact that P is complete. $\square$

Theorem 3

(1) If $P_1, \ldots, P_n, Q_1, \ldots, Q_m \in \mathrm{CQ}$, then

$$\bigvee_i P_i \sqsubseteq \bigvee_j Q_j \;\; \textit{iff} \;\; \forall i \exists j (P_i \sqsubseteq Q_j) \;\; \textit{iff} \;\; \forall i \exists j (Q_j \hookrightarrow P_i).$$

(2) If $P_1, \ldots, P_n, Q_1, \ldots, Q_m \in \mathrm{CQ}^{\neg}$ and $P_1, \ldots, P_n$ are complete, then

$$\bigvee_i P_i \sqsubseteq \bigvee_j Q_i \;\; \textit{iff} \;\; \forall i \exists j (P_i \sqsubseteq Q_j) \;\; \textit{iff} \;\; \forall i \exists j (Q_j \hookrightarrow P_i).$$

Proof. Assume $P_1, \ldots, P_n, Q_1, \ldots, Q_m \in \mathrm{CQ}$. Set $P := \bigvee_i P_i$ and $Q := \bigvee_j Q_j$. If $P \sqsubseteq Q$, then $[P_i] \models P$ so $[P_i] \models Q$. Therefore, for some j, $[P_i] \models Q_j$ and, by Lemma 3, $Q_j \hookrightarrow P_i$. By Theorem 2, $Q_j \sqsubseteq P_i$. The other direction is obvious. If $P_1, \ldots, P_n, Q_1, \ldots, Q_m \in \mathrm{CQ}^{\neg}$, then the same proof works, except that we need the fact that $P_1, \ldots, P_n$ are complete. $\square$

Definition 5 [Completion] The *completion* of Q, which we write $\mathrm{comp}(Q)$, is the maximal disjunction $\bigvee_i Q_i$ of complete non-equivalent queries $Q_i \sqsubseteq Q$, where $Q_i \in \mathrm{CQ}^{\neg}$ and $\mathrm{vars}(Q_i) \subseteq \mathrm{vars}(Q)$.

That is, $\mathrm{comp}(Q)$ is the query (defined only up to order of conjuncts and disjuncts) $\bigvee_{1 \leqslant i \leqslant n} Q_i$ where n the maximal integer such that for each i the following hold:

- for each j, $j \neq i$, Q_i is not equivalent to Q_j,
- Q_i is satisfiable,
- Q_i has the same head as Q,
- the body of Q_i includes all the literals of Q,
- for each k, each k-ary relation R in the schema and each k-tuple $\bar{x}$ of terms from Q either $R(\bar{x})$ or $\neg R(\bar{x})$ occurs in the body of Q_i.

The following is immediate from the definition of comp.

Lemma 4

(1) $\mathrm{comp}(Q) \equiv Q$.
(2) $\mathrm{comp}(Q)$ *is complete.*
(3) If Q *is complete, then* $\mathrm{comp}(Q) = Q$ *(up to order)*
(4) For every schema τ*, there is a polynomial* p_τ *such that if* $\mathrm{comp}(Q) = \bigvee_i Q_i$ *with* $Q_i \in \mathrm{CQ}^\neg$*, then, for all* i*,* $|Q_i| \leqslant p_\tau(|Q|)$.
(5) For every schema τ'*,* $\mathrm{comp}(Q)|_{\tau'} \equiv Q|_{\tau'}$.

Proof.

(1) It follows from the definition that $\mathrm{comp}(Q) \sqsubseteq Q$. Now assume, to get a contradiction, that there is $D\bar{d}$ such that $D\bar{d} \models Q$ and $D\bar{d} \not\models \mathrm{comp}(Q)$. Pick $P \in \mathrm{CQ}^\neg$ to be complete and to satisfy $[P] = D\bar{d}$. Then $[P] \models Q$ and therefore, by Lemma 3, $Q \hookrightarrow P$. By Theorem 2, $P \sqsubseteq Q$ and therefore P must be a disjunct of $\mathrm{comp}(Q)$, contradicting $[P] \not\models \mathrm{comp}(Q)$. It follows that $Q \sqsubseteq \mathrm{comp}(Q)$.
(2) Immediate from the definition of $\mathrm{comp}(Q)$.
(3) This follows from the fact that two complete queries $P, Q \in \mathrm{CQ}^\neg$ are equivalent iff they are equal (up to order).
(4) This follows from the fact that there are at most $2v^r$ literals with v variables and a relational symbol of arity r and therefore at most $2|\tau|v^r$ literals with v variables over τ where r is the maximal arity of a relational symbol in τ.
(5) This follows from 1, 2, and the fact that $\mathrm{comp}(Q)_{\tau'}$ is complete over τ'.

□

Notice that while each Q_i is bounded in size by a polynomial p_τ, the number of such queries Q_i is exponential in the number of variables in Q. Lemma 4 and Theorem 3 imply the following.

Theorem 4 *If* $\mathrm{comp}(P) = \bigvee_i P_i$ *with* $P_1, \ldots, P_n \in \mathrm{CQ}^\neg$ *and* $Q = \bigvee_j Q_m$ *with* $Q_1, \ldots, Q_j \in \mathrm{CQ}^\neg$*, then*

$$\bigvee_i P_i \sqsubseteq \bigvee_j Q_i \;\; \textit{iff} \;\; \forall i \exists j (P_i \sqsubseteq Q_j) \;\; \textit{iff} \;\; \forall i \exists j (Q_j \hookrightarrow P_i).$$

The upper bound for $\mathrm{CQ}^\neg$ and $\mathrm{UCQ}^\neg$ in Theorem 1 follows from Lemma 4 and Theorem 4.

Corollary 1 *If* Q *is over the schema* τ*, then* $P \sqsubseteq Q$ *iff* $P|_\tau \sqsubseteq Q$.

Proof. One direction is clear. For the other direction, assume $P \sqsubseteq Q$ and

$Q = \bigvee_i Q_i$ with $Q_1, \ldots, Q_n \in \mathrm{CQ}^\neg$. Then

$$\begin{aligned} \mathrm{comp}(P) &\sqsubseteq Q && \text{Since } P \equiv \mathrm{comp}(P) \text{ by Lemma 4} \\ \mathrm{comp}(P)|_\tau &\sqsubseteq Q && \text{By Theorem 4 (*)} \\ P|_\tau &\sqsubseteq Q && \text{Since } \mathrm{comp}(P)|_\tau \equiv P|_\tau \text{ by Lemma 4} \end{aligned}$$

(*) since homomorphisms from Q_i must map into τ-literals. □

2.4 Access Patterns

This section contains a review of known results from [20,21,17]. We include most of the proofs in the appendix. An *access pattern* for a k-ary relation R is an expression R^α where α is a word of length k over the alphabet $\{\mathrm{i}, \mathrm{o}\}$. 'i' denotes a required *input slot* and 'o' denotes an *output slot* (no value required). Given access patterns $\mathcal{P}$, an *annotation* of Q assigns to each occurrence of a relation symbol a pattern from $\mathcal{P}$.

Definition 6 [Executable] Q is *executable* if it can be annotated so that every variable of a rule appears first [3] positively in an output slot in the body of that rule.

Definition 7 [Feasible] Q is *feasible* if it is equivalent to an executable query Q'. FEASIBLE($\mathcal{L}$) is the decision problem: for $Q \in \mathcal{L}$, determine whether Q is feasible.

Remark. A set of access patterns is additional information about a schema (the one specifying the base relations or the one consisting of the view heads). Access patterns are necessary to determine whether a query is executable or feasible, and to determine the answerable part of a query (defined below). All other notions defined in this paper (e.g. safety) do not depend on the access patterns.

Theorem 21 in Section 9 below shows that a query Q is feasible iff it is answerable in the intuitive sense. That is, if there is an algorithm which computes Q respecting the access patterns.

For $Q \in \mathrm{CQ}^\neg$, we say that a literal $\ell(\bar{x})$ (not necessarily in Q) is Q-*answerable* if there is an executable $Q' \in \mathrm{CQ}^\neg$ which is a conjunction of $\ell(\bar{x})$ and literals in Q. The *answerable part* of a query Q is another query $\mathrm{ans}(Q)$ defined below. $\mathrm{ans}(Q)$ may be undefined for some queries Q, but when defined it is executable.

Definition 8 [Answerable Part] If $Q \in \mathrm{CQ}^\neg$ is unsatisfiable we set the body

[3] when reading the program that defines it from left to right

of ans(Q) to `false`; otherwise we set the body of ans(Q) to the conjunction of the Q-answerable literals $\{L_1, \ldots, L_m\}$ in Q. in the order $L_1, \ldots, L_m$ where $L_1, \ldots, L_m$ satisfy the following conditions. Set Q_i to be the conjunction of literals $L_1, \ldots, L_i$ or empty in case $i = 0$. We require L_i to be Q_{i-1}-answerable. Notice that if L_i is Q-answerable, then it must be Q_j answerable for some j (this follows from the definition of Q-answerable). Furthermore, if both L_i and L_j for $i < j$ are Q_{i-1}-answerable, we require L_i to appear first [4] in Q. We set head(ans(Q)) := head(Q). However, if the resulting query ans(Q) is unsafe, we say that ans(Q) is undefined. If $Q = \bigvee_i Q_i$ with $Q_1, \ldots, Q_n \in \text{CQ}^\neg$, we set ans($Q$) := $\bigvee_i$ ans(Q_i). In this case ans(Q) is defined iff every ans(Q_i) is defined. For an example, see the last part of Example 1.

The main results on testing feasibility for UCQ$^\neg$ queries are [20]: if defined, ans(Q) is the minimal (under containment) executable query containing Q (Theorem 5); checking feasibility of UCQ$^\neg$ queries can be reduced to checking UCQ$^\neg$ query containment (Corollary 2), and is in fact as hard as checking query containment of UCQ$^\neg$ queries (Theorem 6). Checking feasibility of UCQ queries is **NP**-complete in the size of the query (Theorem 7) [21,17].

Lemma 5 ans(Q) *can be computed in quadratic time in the size of* Q.

Proof. We consider the case when $Q \in \text{CQ}^\neg$; the case $Q \in \text{UCQ}^\neg$ is handled the same way, one rule at a time. Give ans(Q) the same head as Q and build its body one literal at a time as follows. Start with $\mathcal{B}$, the set of bound variables, empty. Find the first literal $\ell(\bar{x})$ in Q not yet added to ans(Q) such that,

- $\ell(\bar{x})$ is positive and there is some access pattern for it in $\mathcal{P}$ such that all variables in $\bar{x}$ which appear in input slots in $\ell(\bar{x})$ are in $\mathcal{B}$, or
- $\ell(\bar{x})$ is negative and its variables are in $\mathcal{B}$.

If there is no such literal, stop. Otherwise, add $\ell(\bar{x})$ to ans(Q), set $\mathcal{B} := \mathcal{B} \cup \{\bar{x}\}$, and repeat. Clearly, this algorithm adds to the body of ans(Q) all the Q-answerable literals in Q and no others and their order satisfies the definition. □

Lemma 6 *If* $Q, E \in \text{CQ}^\neg$, Q *is complete,* ans(E) *is defined, and* $Q \sqsubseteq$ ans(E), *then* ans(Q) *is defined and* ans(Q) $\sqsubseteq$ ans(E).

Lemma 7 *If* ans(Q) *is defined, then* ans(comp(Q)) *is defined and* ans(Q) $\equiv$ ans(comp(Q)).

Lemma 8 *If* ans(Q) *is defined, then* $Q \sqsubseteq$ ans(Q).

Theorem 5 *If* $Q \sqsubseteq E$ *and* E *is executable then* ans(Q) *is defined and*

[4] when reading Q from left to right

$Q \sqsubseteq \text{ans}(Q) \sqsubseteq E$. *[20]*

Corollary 2 *Q is feasible iff* $\text{ans}(Q)$ *is defined and* $\text{ans}(Q) \sqsubseteq Q$.

We write $A \leqslant_m^{\text{P}} B$ if problem A is polynomial time reducible to B by a many-one reduction and $A \equiv_m^{\text{P}} B$ if A is polynomial time equivalent to B. A proof of the following result is included in the Appendix.

Theorem 6 $\text{FEASIBLE}(\text{UCQ}^\neg) \equiv_m^{\text{P}} \text{CONT}(\text{UCQ}^\neg)$ *and therefore is* $\mathbf{\Pi_2^P}$*-complete. [20]*

Theorem 7 $\text{FEASIBLE}(\text{UCQ}) \equiv_m^{\text{P}} \text{CONT}(\text{UCQ})$ *and therefore is* **NP***-complete. [21,17].*

3 An Extension of the Chase

Given a set of constraints, there is a well known procedure for extending a query Q_1 to another query Q_1' by an iterative procedure known as the *chase* so that, for any query Q_2,

$$Q_1 \sqsubseteq_\Sigma Q_2 \text{ iff } Q_1' \sqsubseteq Q_2.$$

The chase was originally introduced to check containment of CQ queries under embedded dependencies [1], but it also applies to query rewriting, as shown in the following sections. In this section, we extend the chase procedure to $\text{UCQ}^\neg$ queries and corresponding sets of constraints.

To any query language $\mathcal{L}$, we associate a class of constraints of the form $\text{IC}(\mathcal{L}) := \{\, \forall \bar{x}\, (U \rightarrow V) \mid U, V \in \mathcal{L}\}$ where $\bar{x}$ is the set of free variables in both U and V. Such constraints express the containment of U in V and are precisely the *embedded dependencies* [1] when $\mathcal{L}$ is the language of conjunctive queries with equality atoms.

Unless otherwise specified, we assume all constraints are subsets of $\text{IC}(\text{UCQ}^\neg)$**. Furthermore** Σ **always denotes a set of** $\text{IC}(\text{UCQ}^\neg)$ **constraints.**

We observe first that constraints in $\text{IC}(\text{UCQ}^\neg)$ can be normalized to eliminate disjunction and existential quantification from the premise of the implication. Indeed, consider the constraint $\sigma \in \text{IC}(\text{UCQ}^\neg)$

$$\sigma: \quad \forall \bar{x}\ \varphi(\bar{x}) \rightarrow \xi(\bar{x})$$

where $\varphi, \xi \in \text{UCQ}^\neg$ with $\text{free}(\varphi) = \text{free}(\xi) = \bar{x}$ and $\varphi(\bar{x}) = \bigvee_{i=1}^n \varphi_i(\bar{x})$ with

$\varphi_i \in \mathrm{CQ}^\neg$ for all $1 \leqslant i \leqslant n$. Then it follows from application of DeMorgan's laws that an instance satisfies σ if and only if it satisfies the set of constraints $\{\sigma_i\}_{i=1}^n$:

$$\sigma_i : \quad \forall \bar{x}\ \varphi_i(\bar{x}) \rightarrow \xi(\bar{x}).$$

Also notice that, denoting $\bar{z} := \mathrm{vars}(\varphi_i) \setminus \bar{x}$, σ_i is equivalent (again by DeMorgan) to

$$\forall \bar{x} \forall \bar{z}\ \varphi_i(\bar{x}, \bar{z}) \rightarrow \xi(\bar{x}).$$

In the remainder of this section we assume that all constraints are normalized to the form (11) below and we define the chase for such constraints only.

Let $\sigma \in \mathrm{IC}(\mathrm{UCQ}^\neg)$ be a normalized integrity constraint of the form

$$\sigma : \quad \forall \bar{x}\ \psi(\bar{x}) \rightarrow \bigvee_{i=1}^{l} \exists \bar{y}_i\ \xi_i(\bar{x}, \bar{y}_i) \tag{11}$$

where ψ is a quantifier-free $\mathrm{CQ}^\neg$ with $\bar{x} = \mathrm{vars}(\psi)$, and for each i, ξ_i is a quantifier-free $\mathrm{CQ}^\neg$ with $\{\bar{y}_i\} \subseteq \{\mathrm{vars}(\xi_i)\} \subseteq \{\bar{x}\} \cup \{\bar{y}_i\}$.

Chase Step. Let $Q \in \mathrm{CQ}^\neg$ and assume w.l.o.g. that $\mathrm{vars}(Q) \cap \mathrm{vars}(\sigma) = \emptyset$ and that $\forall i \neq j\ \bar{y}_i \cap \bar{y}_j = \emptyset$ (this is always achievable through variable renaming).

We say that a *chase step* of Q with σ *applies* iff there is a homomorphism h from ψ to Q, both viewed as boolean queries, such that for each i, h has no extension to a homomorphism from $\psi \wedge \xi_i$ to ψ. In other words, there is no homomorphism h' from $\psi \wedge \xi_i$ to ψ such that $h'(\bar{x}) = h(\bar{x})$. The *result* of this chase step, denoted $\mathrm{step}(Q, \sigma, h)$, is a $\mathrm{UCQ}^\neg$ query obtained as follows. First construct the disjunction $\bigvee_{i=1}^{l} Q \wedge h'(\xi_i)$, where h' is a mapping on $\mathrm{vars}(\sigma)$ that extends h to be the identity on $\bar{y}_i$. Next, remove all unsatisfiable $\mathrm{CQ}^\neg$s. Note that all $\mathrm{CQ}^\neg$s may be unsatisfiable in which case the result of the chase step is the unsatisfiable empty disjunction `false`.

Example 4 Consider the constraint

$$\forall x \forall y\ R(x, y) \rightarrow \exists z\ S(x, z) \wedge \neg E(z, x) \vee T(y, x) \tag{12}$$

Then no chase step with (12) applies to the boolean query $Q :- R(m, n) \wedge T(n, m)$ because the only homomorphism $h = \{x \mapsto m, y \mapsto n\}$ from $R(x, y)$ to $R(m, n)$ is its own extension to $R(x, y) \wedge T(y, x)$ and therefore since one of the disjunts of the conclusion is satisfied, the whole conclusion is satisfied.

However, a chase step applies to $Q :- R(m,n)$, yielding

$$U :- R(m,n) \wedge S(m,z) \wedge \neg E(z,m)$$
$$U :- R(m,n) \wedge T(n,m)$$

Note that no unsatisfiable disjuncts were created in this case.

In contrast, a chase step of $Q :- R(m,n) \wedge \neg T(n,m)$ yields

$$U :- R(m,n) \wedge \neg T(n,m) \wedge S(m,z) \wedge \neg E(z,m)$$

since $R(m,n) \wedge T(n,m) \wedge \neg T(n,m)$ is unsatisfiable. $\square$

We lift the definition of chase step of a $\text{CQ}^\neg$ to that of $\text{UCQ}^\neg$s. Let $Q = \bigvee_j Q_j$, and let $\sigma \in \text{IC}(\text{UCQ}^\neg)$ such that w.l.o.g. $\text{vars}(\sigma) \cap \text{vars}(Q) = \emptyset$. Then a chase step of Q with σ applies iff there is a j_0 such that a chase step with σ applies to Q_{j_0} using some homomorphism h. The result of the chase step on Q is defined as $\bigvee_{j \neq j_0} Q_j \vee \text{step}(Q_{j_0}, \sigma, h)$.

Deterministic Chase Sequence. Let $\Sigma \subseteq \text{IC}(\text{UCQ}^\neg)$ be a set of constraints. A *chase sequence* of a query Q_0 with Σ is a sequence of queries $Q_0, Q_1, \ldots, Q_n$ such that Q_{i+1} is the result of applying a chase step with some $\sigma \in \Sigma$ to Q_i. Note that for each Q_i several chase steps may be simultaneously applicable, thus potentially allowing several distinct chase sequences of Q_0 with Σ. However, given some arbitrary total order O on Σ, O uniquely determines a chase sequence as follows. As long as chase steps apply, they are executed. Whenever several chase steps apply simultaneously to Q_i, we apply a step corresponding to the minimal constraint in the order O, say σ_m. Note that even for σ_m several chase steps may apply, because there are several homomorphisms from $\text{vars}(\sigma_m)$ to the variables of Q_i. In that case, execute the step corresponding to the homomorphism h which minimizes $h(\text{vars}(\sigma_m))$ w.r.t. the lexicographic order of Q_i's variables. We say that the chase determined by O *terminates* if the corresponding chase sequence has finite length n. We define the *chase result* of Q_0 as Q_n and denote it as $\text{chase}(Q_0, \Sigma, O)$.

The chase does not always terminate (even in the case with no negation) and its syntactic form depends on the order O. However if the chase terminates for any two orders O_1 and O_2, then the two chase results are equivalent: $\text{chase}(Q, \Sigma, O_1) \equiv \text{chase}(Q, \Sigma, O_2)$ which is the same as saying that they are homomorphically equivalent. That is, there are homomorphisms

$$\text{chase}(Q, \Sigma, O_1) \hookrightarrow \text{chase}(Q, \Sigma, O_2) \text{ and } \text{chase}(Q, \Sigma, O_2) \hookrightarrow \text{chase}(Q, \Sigma, O_1).$$

All our results below which depend on the chase result hold under homomor-

phic equivalence and therefore we do not specify any specific chase order in our results or algorithms. Any order which leads to chase termination is sufficient and later we will consider conditions which ensure that the chase terminates for any order.

We therefore define the result of the chase up to equivalence. To this end, we introduce Negation Constraints:

Definition 9 [Negation Constraints] $\Sigma^\tau_\neg \subseteq \mathrm{IC}(\mathrm{UCQ}^\neg)$ is the smallest set of constraints which contains, for each k, each k-ary relation R in the schema τ and some k-tuple of distinct $\bar{x}$ of variables, the constraint $\forall \bar{x}\,(\mathtt{true} \rightarrow (\, R(\bar{x}) \vee \neg R(\bar{x})\,)\,)$.

We allow queries which are unsafe in the conclusion of the constraints (we need them for $\Sigma^\tau_\neg$); however if Q is safe, then $\mathrm{chase}(Q, \Sigma, O)$ is also safe, even when Σ includes unsafe sentences.

Definition 10 [Chase Result Q^Σ] $Q^\Sigma := \mathrm{chase}(Q, \Sigma \cup \Sigma^\tau_\neg, O)$ for some order on which the chase terminates (if there is such order).

We write $Q^{\Sigma,\Sigma'}$ for $(Q^\Sigma)^{\Sigma'}$ which in general is not equivalent to $Q^{\Sigma\cup\Sigma'}$.

In the following, we use the notion of chase result Q^Σ to extend previous results which do not handle negation. In particular, we reduce containment under constraints to containment over all databases. While the exact statement of this reduction is given in Theorems 8 and 9 below, they essentially state that

$$Q_1 \sqsubseteq_\Sigma Q_2 \text{ iff } Q_1^\Sigma \sqsubseteq Q_2.$$

Previous work shows the reduction for CQ query containment under IC(CQ) constraints [1], and UCQ queries under IC(UCQ) [5]. We extend these to $\mathrm{UCQ}^\neg$ queries and $\mathrm{IC}(\mathrm{UCQ}^\neg)$ constraints next.

Theorem 8 states that the chase preserves equivalence to the original query under the constraints.

Theorem 8 (Soundness of the Chase)

(1) If a chase step of query Q with constraint $\sigma \in \Sigma$ applies using homomorphism h, then $Q \equiv_\Sigma \mathrm{step}(Q, \sigma, h)$.
(2) Let O be any total order on Σ which determines a terminating chase of Q. Then $Q \equiv_\Sigma \mathrm{chase}(Q, \Sigma, O)$.

The proof is given in the Appendix.

Theorem 9 (Completeness of the Chase) *If* $\mathrm{chase}(P, \Sigma^\tau_\neg \cup \Sigma, O)$ *termi-*

nates for some O, then

$$P \sqsubseteq_\Sigma Q \textit{ iff } \text{chase}(P, \Sigma^\tau_\neg \cup \Sigma, O) \sqsubseteq Q$$

The proof is given in the Appendix.

It is well-known that checking termination of the chase is undecidable even for the constraint language IC(CQ) [1]. [5] and [8] introduce a sufficient condition, checkable in **P**, for termination of the chase with IC(UCQ) constraints. It is fairly wide and generalizes the notions of full and acyclic dependencies [1]. The condition requires a set of constraints to have *stratified witnesses*.[5] Here we extend the notion to sets of IC(UCQ$^\neg$) constraints.

Definition 11 [Chase Flow Graph[6]] Given $\Sigma \subseteq$ IC(UCQ$^\neg$) define its *chase flow graph* $G = (V, E)$ as a directed graph whose edge labels can be either $\forall$ or $\exists$. G is constructed as follows: for every relation R of arity a mentioned in Σ, V contains a node R^i $(1 \leqslant i \leqslant a)$. For every pair of relations R, S of arities a, a' if $R(\bar{x})$ appears in the premise and $S(\bar{y})$ in the conclusion of some constraint $\sigma \in \Sigma$,

- if $x_i = y_j$, then add a $\forall$-labeled edge from R^i to S^j and
- if y_j is existentially quantified, then add an $\exists$-labeled edge from R^i to S^j.

Notice that this definition is independent of whether R and S appear within the scope of negation or disjunction.

Definition 12 [Set of Constraints with Stratified Witnesses] We say that a set of constraints has *stratified witnesses* if it has no cycles through $\exists$-edges.

We introduce the following notations.

- u denotes the maximum number of universally quantified variables mentioned in a disjunct of a constraint's conclusion:

$$u := \max\{|\text{vars}(\xi_i) \cap \{\bar{x}\}| \ : \ (\forall \bar{x} \psi \rightarrow \bigvee_i \exists \bar{y}_i \xi_i) \in \Sigma\}$$

- Given a constraint $\sigma \in \Sigma$, we define e_σ denotes the maximum number of existentially quantified variables mentioned in a disjunct of a constraint's conclusion: $e_\sigma := \max_i |\bar{y}_i|$ where $\sigma = \forall \bar{x} \psi \rightarrow \bigvee_i \exists \bar{y}_i \xi_i$. We set $e := \sum_{\sigma \in \Sigma} e_\sigma$.

[5] The notion first arose in a conversation between the first author and Lucian Popa. It was then independently used in [5] and in [8] (in the latter paper, under the term *weakly acyclic*).

[6] The chase flow graph is similar to the graph used to determine the existence of stratified normal forms for ILOG programs [12]. These invent object identities, just like the chase invents new variables.

According to the following result, sets of constraints with stratified witnesses enjoy the important property that the chase with them is guaranteed to terminate, and moreover to produce a result which is a union of $\text{CQ}^\neg$ queries, such that there are at most exponentially many union members, each of size polynomial in the size of the original query.

Theorem 10 *(Chase Termination)*

(1) For any $Q \in \text{UCQ}^\neg$, any $\Sigma \subseteq \text{IC}(\text{UCQ}^\neg)$ with stratified witnesses and any total order O on Σ, the chase terminates.
(2) Moreover, assume that $\text{chase}(Q, \Sigma, O) := \bigvee_i Q_i$ and denote with l the maximum number of $\exists$-edges on any path in the chase flow graph of Σ. Then for each i,

$$|\text{vars}(Q_i)| \leq (1+e)^{u^l} |\text{vars}(Q)|^{u^l}$$

where e and u are as defined above.

The proof is given in the Appendix.

Corollary 3 *Under the hypothesis of Theorem 10, fix the database schema and Σ. Then the size of each $Q_i \in \text{CQ}^\neg$ in the chase result is polynomial in the size of Q.*

Remark. The chase with $\Sigma^\tau_\neg$ gives us a procedure for computing the completion of a query.

Proposition 1 *For any $Q \in \text{UCQ}^\neg$ and any total order O on $\Sigma^\tau_\neg$, the corresponding chase terminates and $\text{comp}(Q) = \text{chase}(Q, \Sigma^\tau_\neg, O)$ up to order of conjuncts and disjuncts.*

4 Integrity Constraints

We consider $\{Q, \mathcal{P}, \Sigma\}$, i.e., the problem of answering a query Q in the presence of access patterns $\mathcal{P}$ and integrity constraints Σ.

Definition 13 [Σ-Feasible] Q is Σ-*feasible* if it is Σ-equivalent to an executable query Q'. $\text{FEASIBLE}_\Sigma(\mathcal{L})$ is the decision problem: for $Q \in \mathcal{L}$ and fixed Σ, decide whether Q is Σ-feasible.

The main results in this section are that, if defined, $\text{ans}(Q^\Sigma)$ is the minimal (under Σ-containment) executable query Σ-containing Q (Theorem 11), that checking Σ-feasibility of $\text{UCQ}^\neg$ queries can be reduced to checking containment

of UCQ$^\neg$ queries (Corollary 4), and that in those cases where Q^Σ is well-defined (i.e., the chase terminates) and not too large its complexity is the same as that of checking containment of UCQ$^\neg$ queries (Theorem 12b). Corresponding results hold for CQ, CQ$^\neg$, and UCQ (Theorem 12a). We outline the algorithms REWRITE and FEASIBLE which use the following functions:

- ans(Q), which given a query Q, produces the query ans(Q). A quadratic time algorithm for this function is outlined in the proof of Lemma 5.
- chase(Q, Σ, O), which given a query Q, a set of constraints Σ, and an order on the constraints O, produces the query chase(Q, Σ, O) as described in Section 3. In general, no guarantees are given for the running time or space of chase(Q, Σ, O); in fact, it may not even terminate (unless Σ has stratified witnesses).
- contained(P, Q), which given queries P and Q, returns true if $P \sqsubseteq Q$, false otherwise (its complexity is given in Theorem 1).

Note that algorithm REWRITE(Q, Σ) may return `undefined` or may not terminate; similarly, FEASIBLE(Q, Σ) may not terminate. Theorem 11 and Corollary 4 below show that algorithms REWRITE and FEASIBLE are correct and complete regardless of the chase order O, as long as the chase terminates.

function REWRITE(Q, Σ)

(1) Compute $\Sigma' := \Sigma \cup \Sigma^\tau_\neg$ and pick some order O for Σ';

(2) $Q^1 :=$ chase(Q, Σ', O);

(3) $Q^2 :=$ ans(Q^1);

(4) **return** Q^2.

Here we give a simplified version of FEASIBLE which gives an exponential time algorithm. This algorithm can be parallelized to give a $\mathbf{\Pi_2^P}$ algorithm when Q and Σ satisfy the assumptions of Theorem 12, as outlined in the proof of that theorem.

function FEASIBLE(Q, Σ)

(1-3) *same as (1–3) of* REWRITE(Q, Σ);

(4) **if** $Q^2 =$ `undefined` **then return** `false`;

(5) $Q^3 :=$ chase(Q^2, Σ', O);

(6) **return** contained(Q^3, Q).

The following shows that, if it is at all possible to overestimate Q via an executable query, then the query returned by algorithm REWRITE is a *minimally* Σ-containing rewriting of Q, i.e. it is the executable rewriting of Q which least

overestimates Q.

Theorem 11 *If $Q \sqsubseteq_\Sigma E$, E executable, and Q^Σ is defined, then* $\text{ans}(Q^\Sigma)$ *is defined and $Q \sqsubseteq_\Sigma \text{ans}(Q^\Sigma) \sqsubseteq E$.*

Proof. Assume $Q \sqsubseteq_\Sigma E$, E executable, and Q^Σ is defined. Then by Theorem 9, $Q^\Sigma \sqsubseteq E$. Thus, by Theorem 5, $\text{ans}(Q^\Sigma)$ is defined and $Q^\Sigma \sqsubseteq \text{ans}(Q^\Sigma) \sqsubseteq E$. By Theorem 9, $Q \sqsubseteq_\Sigma \text{ans}(Q^\Sigma) \sqsubseteq E$. □

According to the following corollary of Theorem 11, algorithm FEASIBLE is a decision procedure for Σ-feasibility provided that the chase terminates.

Corollary 4 *The following are equivalent:*

(1) Q is Σ-feasible and Q^Σ is defined.
(2) $\text{ans}(Q^\Sigma)$ *is defined and* $\text{ans}(Q^\Sigma) \sqsubseteq_\Sigma Q$.
(3) $\text{ans}(Q^\Sigma)^\Sigma$ *is defined and* $\text{ans}(Q^\Sigma)^\Sigma \sqsubseteq Q$.

Proof. If (1) holds, then $Q \sqsubseteq_\Sigma E$ for some executable E and therefore (2) holds by Theorem 11. If (2) holds, then certainly $\text{ans}(Q^\Sigma)^\Sigma$ is defined (since the chase Q^Σ terminates) and $\text{ans}(Q^\Sigma)^\Sigma \sqsubseteq Q$ holds by Theorem 9, so (3) holds. If (3) holds, then certainly Q^Σ is defined and $\text{ans}(Q^\Sigma)$ is executable and, by Theorem 9, Σ-contained in Q, so Q is Σ-feasible by Corollary 2. □

The following result provides a fairly general condition on the behavior of the chase which implies (tight) upper bounds for deciding Σ-feasibility.

Theorem 12

(1) If $\Sigma \subseteq \text{IC(UCQ)}$ and there is a polynomial p such that for all Q, $Q^\Sigma \in \text{UCQ}$ is defined and $|Q^\Sigma| \leqslant p(|Q|)$, then $\text{FEASIBLE}_\Sigma(\text{UCQ})$ *is* **NP**-*complete.*
(2) If $\Sigma \subseteq \text{IC}(\text{UCQ}^\neg)$ and there is a polynomial p such that for all Q, $Q^\Sigma (= \bigvee_i Q'_i) \in \text{UCQ}^\neg$ is defined and for all i, $|Q'_i| \leqslant p(|Q|)$, then $\text{FEASIBLE}_\Sigma(\text{UCQ}^\neg)$ *is* $\mathbf{\Pi_2^P}$-*complete.*

Proof.

(1) Given $Q \in \text{UCQ}$ and Σ and p as in the hypotheses, compute $\text{ans}(Q^\Sigma)^\Sigma$ (if this is undefined return 'no') and check whether $\text{ans}(Q^\Sigma)^\Sigma \sqsubseteq Q$. Since $|\text{ans}(Q^\Sigma)^\Sigma| \leqslant |Q^\Sigma| \leqslant p(|Q|)$ the latter can be checked in **NP** in the size of Q. The lower bound follows from the fact that we can reduce FEASIBLE(UCQ) to $\text{FEASIBLE}_\Sigma(\text{UCQ})$ by taking $\Sigma = \emptyset$.
(2) Assume the hypotheses of the theorem, $Q = \bigvee_j Q_j$, and $\text{ans}(Q^\Sigma)^\Sigma = \bigvee_\ell Q''_\ell$ with $Q_j, Q''_\ell \in \text{CQ}^\neg$. Since each Q''_ℓ is complete, by Theorem 3, we can test whether $\text{ans}(Q^\Sigma)^\Sigma \sqsubseteq Q$ (if the left hand side is undefined return

'no') by testing $\forall \ell \exists j (Q''_\ell \sqsubseteq Q_j)$. To test the latter in $\mathbf{\Pi_2^P}$ in the size of Q proceed as follows.

- $\forall$-verify that for every "candidate" $P, P' \in \mathrm{CQ}^\neg$ such that P is a disjunct of Q^Σ and P' is a disjunct of $\mathrm{ans}(P)^\Sigma$ (here P' takes the place of Q''_ℓ)
- $\exists$-guess j and a homomorphism $Q_j \to P'$ and verify the homomorphism in polynomial time.

The important points are that

- We can check whether P is a disjunct of Q^Σ in **NP** time in $|P|$. In particular, we don't need to generate all of Q^Σ; there is a chase sequence $Q = Q^0, Q^1, \ldots, Q^n = Q^\Sigma$ (with Q^{s+1} obtained by a chase step on $Q^s_{j_0(s)}$) and it is enough to find a sequence of $\mathrm{CQ}^\neg$ queries $P^0, P^1, P^2, \ldots, P^n$ such that $P^s = Q^s_{j_0(s)}$ and $P^n = P$.
- Similarly, we can check whether P' is a disjunct of $\mathrm{ans}(P)^\Sigma$ in **NP** in $|P'|$ and $|\mathrm{ans}(P)|$.
- Since we have assumed that for all i, $|Q'_i| \leqslant p(|Q|)$, we know that if P and P' satisfy the conditions above, then $|P|, |P'| \leqslant p(|Q|)$.

The lower bound follows from the fact that we can reduce $\mathrm{FEASIBLE}(\mathrm{UCQ}^\neg)$ to $\mathrm{FEASIBLE}_\Sigma(\mathrm{UCQ}^\neg)$ by taking $\Sigma = \emptyset$.

□

The fact that Q^Σ is defined only up to equivalence is not a concern for our needs, due to the following result, which implies that $\mathrm{ans}(Q^\Sigma)$ is also defined up to equivalence.

Lemma 9

(1) If $P \sqsubseteq Q$ *and* $\mathrm{ans}(Q)$ *is defined, then* $\mathrm{ans}(P)$ *is defined and* $\mathrm{ans}(P) \sqsubseteq \mathrm{ans}(Q)$.

(2) If $P \equiv Q$ *and* $\mathrm{ans}(Q)$ *is defined, then* $\mathrm{ans}(P)$ *is defined and* $\mathrm{ans}(P) \equiv \mathrm{ans}(Q)$.

Proof. (1) If $\mathrm{ans}(Q)$ is defined then it is executable and $P \sqsubseteq Q \sqsubseteq \mathrm{ans}(Q)$. By Theorem 5, $\mathrm{ans}(P)$ is defined and $\mathrm{ans}(P) \sqsubseteq \mathrm{ans}(Q)$. (2) follows from (a). □

Theorem 10 and Theorem 12 immediately imply the following:

Corollary 5 *For any fixed* $\Sigma \subseteq \mathrm{IC}(\mathrm{UCQ}^\neg)$ *with stratified witnesses* $\mathrm{FEASIBLE}_\Sigma(Q^\Sigma)$ *is* $\mathbf{\Pi_2^P}$*-complete in the size of* Q.

5 Views

We now consider the problem $\{Q, \mathcal{V}, \mathcal{P}, \Sigma_c\}$: given

(1) a query Q over schema τ,
(2) a set of views $\mathcal{V}$ given as UCQ$^\neg$ queries $V_1, \ldots, V_n$ over schema τ
(3) a set of access patterns $\mathcal{P}$ over schema τ_V, where
$\tau_V := \{\text{head}(V_1), \ldots, \text{head}(V_n)\}$, and
(4) a set of constraints Σ_c over τ,

we are interested in finding a query E such that

(1) E is over schema τ_V,
(2) E is executable w.r.t. the access patterns $\mathcal{P}$, and
(3) E is Σ-equivalent to Q,

where $\Sigma = \Sigma_c \cup \Sigma_f^{\mathcal{V}} \cup \Sigma_b^{\mathcal{V}}$, and $\Sigma_f^{\mathcal{V}}, \Sigma_b^{\mathcal{V}} \subseteq \text{IC}(\text{UCQ}^\neg)$ are "forward" and "backward" constraints over $\tau \cup \tau_V$ which capture the semantics of the views as follows:

$$\Sigma_f^{\mathcal{V}} := \{\forall \bar{x}_i \bar{y}_i \, (\text{body}(V_i) \rightarrow \text{head}(V_i)) \mid V_i \in \mathcal{V}, 1 \leqslant i \leqslant n\}$$

$$\Sigma_b^{\mathcal{V}} := \{\forall \bar{x}_i \, (\text{head}(V_i) \rightarrow \exists \bar{y}_i \text{body}(V_i)) \mid V_i \in \mathcal{V}, 1 \leqslant i \leqslant n\}$$

where $\bar{x}_i$ are the variables in $\text{head}(V_i)$, and $\bar{y}_i$ are the variables in $\text{body}(V_i)$ which do not appear in $\text{head}(V_i)$. We call such E an *executable* Σ_c*-rewriting* of Q using $\mathcal{V}$.

Definition 14 [$\Sigma_c, \mathcal{V}$-Feasible] Q is $\Sigma_c, \mathcal{V}$*-feasible* if there is an executable Σ_c-rewriting of Q using $\mathcal{V}$. $\text{FEASIBLE}_{\Sigma_c, \mathcal{V}}(\mathcal{L})$ is the decision problem: for $Q \in \mathcal{L}$ and fixed Σ_c and $\mathcal{V}$, decide whether Q is $\Sigma_c, \mathcal{V}$-feasible.

Notice that this provides a reduction of $\{Q, \mathcal{V}, \mathcal{P}, \Sigma_c\}$ to $\{Q, \mathcal{P}, \Sigma'\}$, which we covered in Section 4. In addition to the above reduction, the main results in this section are the following. It is enough to consider $Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}}$ instead of Q^Σ for computing the answerable part (but for testing feasibility we also need $\Sigma_b^{\mathcal{V}}$). If defined, $\text{ans}(Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}} | \tau_V)$ is the minimal (under Σ-containment) executable query over τ_V Σ-containing Q (Theorem 13). We write $Q|_\tau$ for the query with the same head as Q and with body given by the literals in Q which have relation symbols in schema τ. It follows that checking whether there is an executable Σ_c-rewriting of a query Q using $\mathcal{V}$ can be reduced to checking containment (Corollary 6). We also show that we can stratify the chase and that we only need special conditions on Σ_c (but not on $\Sigma_f^{\mathcal{V}}$ or $\Sigma_b^{\mathcal{V}}$) to guarantee that Q^Σ is well-defined and suitably small (Theorem 14). We outline the algorithms ViewRewrite and ViewFeasible which use the functions $\text{ans}(Q)$,

chase(Q, Σ, O), and contained(P, Q).

function ViewRewrite$(Q, \Sigma_c, \mathcal{V})$

(1) Compute $\Sigma_c{}' := \Sigma_c \cup \Sigma_\neg^\tau$ and pick some order O_c for $\Sigma_c{}'$;

(2) Compute $\Sigma_f^{\mathcal{V}'} := \Sigma_f^{\mathcal{V}} \cup \Sigma_\neg^\tau$ and pick some order O_f for $\Sigma_f^{\mathcal{V}'}$;

(3) $Q^1 :=$ chase$(Q, \Sigma_c{}', O_c)$;

(4) $Q^2 :=$ chase$(Q^1, \Sigma_f^{\mathcal{V}'}, O_f)$;

(5) $Q^3 := Q^2|\tau_V$ (that is, drop all τ literals);

(6) $Q^4 :=$ ans(Q^3);

(7) **return** Q^4.

ViewRewrite$(Q, \Sigma_c, \mathcal{V})$ may return `undefined` or may not terminate. Similarly, ViewFeasible$(Q, \Sigma_c, \mathcal{V})$ may not terminate. Theorem 13 and Corollary 6 show that these algorithms are correct and complete in case the chase terminates regardless of the order O. The simplified version of ViewFeasible below results in an exponential time algorithm; however, it can be parallelized to give a $\mathbf{\Pi_2^P}$ (in the size of Q) algorithm when Q and Σ satisfy the assumptions of Theorem 14.

function ViewFeasible$(Q, \Sigma_c, \mathcal{V})$

(1–6) *same as (1–6) of* ViewRewrite$(Q, \Sigma_c, \mathcal{V})$;

(7) **if** $Q^4 =$ `undefined` **then return** `false`;

(8) Compute $\Sigma_b^{\mathcal{V}'} := \Sigma_b^{\mathcal{V}} \cup \Sigma_\neg^\tau$ and pick some order O_b for $\Sigma_b^{\mathcal{V}'}$;

(9) $Q^5 :=$ chase$(Q^4, \Sigma_b^{\mathcal{V}'}, O_b)$;

(10) $Q^6 :=$ chase$(Q^5, \Sigma_c{}', O_c)$;

(11) $Q^7 := Q^6|\tau$ (that is, drop all τ_V literals);

(12) **return** contained(Q^7, Q).

Example 5 We illustrate algorithm ViewRewrite on the setting from Example 2.

Σ_c contains the constraint

$$\forall a \forall t \; C(a,t) \rightarrow R(a,t) \tag{13}$$

$\Sigma_\neg^\tau$ contains

$$\forall a \forall t \, \texttt{true} \rightarrow (R(a,t) \vee \neg R(a,t)) \tag{14}$$
$$\forall a \forall t \, \texttt{true} \rightarrow (C(a,t) \vee \neg C(a,t)), \tag{15}$$

$\Sigma_{\mathrm{f}}^{\mathcal{V}}$ contains

$$\forall a \forall t\ C(a,t) \wedge R(a,t) \rightarrow V_1(a,t) \tag{16}$$
$$\forall a \forall t\ C(a,t) \wedge \neg R(a,t) \rightarrow V_2(a,t), \tag{17}$$

and $\Sigma_{\mathrm{b}}^{\mathcal{V}}$ contains

$$\forall a \forall t\ V_1(a,t) \rightarrow C(a,t) \wedge R(a,t) \tag{18}$$
$$\forall a \forall t\ V_2(a,t) \rightarrow C(a,t) \wedge \neg R(a,t). \tag{19}$$

We are given query $Q(a,t) :- C(a,t)$.

Step 3 of ViewRewrite chases Q with $\Sigma_{\mathrm{c}} \cup \Sigma_{\neg}^{\tau}$. Suppose it first chases Q with constraint (14), to obtain

$$Q'(a,t) :- C(a,t), R(a,t)$$
$$Q'(a,t) :- C(a,t), \neg R(a,t)$$

whose second rule then chases with (13) to

$$Q''(a,t) :- C(a,t), \neg R(a,t), R(a,t)$$

which is unsatisfiable and is dropped from the chase step result. No further chase step applies to the remaining rule, so Step 3 yields

$$Q^1(a,t) :- C(a,t), R(a,t).$$

In Step 4, ViewRewrite chases Q^1 with $\Sigma_{\mathrm{f}}^{\mathcal{V}'}$. Only one chase step applies, namely with (16), yielding

$$Q^2(a,t) :- C(a,t), R(a,t), V_1(a,t).$$

In Step 5, Q^2 is reduced to the atoms mentioning only the view vocabulary $\tau_V = \{V_1, V_2\}$, yielding

$$Q^3(a,t) :- V_1(a,t).$$

Since V_1's access pattern is 'oo', ViewRewrite returns $Q^4 = \mathrm{ans}(Q^3) = Q^3$.

Q^3 turns out to be an equivalent rewriting of Q. Indeed, algorithm ViewFeasible checks this as follows. During the chases in Steps 9 and 10, only one chase step applies, namely with (18), yielding

$$Q^6(a,t) :- V_1(a,t), C(a,t), R(a,t).$$

Then Step 11 drops all view literals, constructing

$$Q^7(a,t) :- C(a,t), R(a,t)$$

which is obviously contained in Q, as witnessed by the identity homomorphism from Q into Q^7. Therefore, algorithm ViewFeasible returns `true`. $\square$

Analogously to Theorem 11, we have that if there is an an executable query expressed in terms of the views containing Q, then the query returned by algorithm ViewRewrite is guaranteed to be the minimal (under Σ-containment) executable overestimate of Q.

Theorem 13 *If $\Sigma = \Sigma_c \cup \Sigma_f^{\mathcal{V}} \cup \Sigma_b^{\mathcal{V}}$, $Q \sqsubseteq_\Sigma E$, E is an executable query over τ_V, and $Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}}$ is defined, then $\mathrm{ans}(Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}}|_{\tau_V})$ is defined and*

$$Q \sqsubseteq_\Sigma \mathrm{ans}(Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}}|_{\tau_V}) \sqsubseteq E.$$

Proof. First notice that $Q^\Sigma = Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}, \Sigma_b^{\mathcal{V}}} = Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}}$ because $\Sigma_f^{\mathcal{V}}$ only introduces atoms with relation symbols from τ_V and these in turn can only "fire" constraints from $\Sigma_b^{\mathcal{V}}$ which reintroduce bodies that have already been matched (with new quantified variables). Such chase steps never apply.

Therefore since $Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}}$ is defined, so is Q^Σ. Then by Theorem 11, $\mathrm{ans}(Q^\Sigma)$ is defined and $Q \sqsubseteq_\Sigma \mathrm{ans}(Q^\Sigma) \sqsubseteq E$. Then by Corollary 1, $Q \sqsubseteq_\Sigma \mathrm{ans}(Q^\Sigma)|_{\tau_V} \sqsubseteq E$. Because all access patterns are over τ_V, we have $\mathrm{ans}(Q^\Sigma|_{\tau_V}) \equiv \mathrm{ans}(Q^\Sigma)|_{\tau_V}$. Therefore, $Q \sqsubseteq_\Sigma \mathrm{ans}(Q^\Sigma|_{\tau_V}) \sqsubseteq E$ as desired.

Finally, since $\mathrm{ans}(Q^\Sigma)$ is defined, $\mathrm{ans}(Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}})$ is defined and therefore $\mathrm{ans}(Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}}|_{\tau_V})$ is defined, again because all access patterns are over τ_V. $\square$

In Corollary 6 part 2 (below), the effect of the chase, of computing the answerable part, and of restricting the result to a subschema can be described as follows:

- In $Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}}$ we introduce the view heads.
- In $\mathrm{ans}(Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}}|_{\tau_V})$ we remove the original literals in Q and the view bodies.
- In $\mathrm{ans}(Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}}|_{\tau_V})^{\Sigma_b^{\mathcal{V}}, \Sigma_c}$ we expand the view heads to again include their bodies which we chase with Σ_c.
- At this point, we have a query over $\tau \cup \tau_V$, but since Q is over τ, only the τ part matters, which is why we use $\mathrm{ans}(Q^{\Sigma_c, \Sigma_f^{\mathcal{V}}}|_{\tau_V})^{\Sigma_b^{\mathcal{V}}, \Sigma_c}|_\tau$.

Corollary 6

(1) There is an executable Σ_c-rewriting of Q using $\mathcal{V}$ iff $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})$ *is defined and* $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V}) \sqsubseteq_\Sigma Q$*, where* $\Sigma = \Sigma_c \cup \Sigma_f^{\mathcal{V}} \cup \Sigma_b^{\mathcal{V}}$.
(2) If $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})^{\Sigma_b^{\mathcal{V}},\Sigma_c}$ *is defined, then there is an executable Σ_c-rewriting of Q using $\mathcal{V}$ iff* $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})^{\Sigma_b^{\mathcal{V}},\Sigma_c}|_\tau \sqsubseteq Q$.

Proof. (1) If there is an executable Σ_c-rewriting E of Q using $\mathcal{V}$, then $E \equiv_\Sigma Q$ and by Theorem 13,

$$\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V}) \sqsubseteq E \equiv_\Sigma Q$$

and $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})$ is defined. Conversely, if $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})$ is defined and $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V}) \sqsubseteq_\Sigma Q$, then since

$$Q \equiv_\Sigma Q^\Sigma \sqsubseteq Q^\Sigma|_{\tau_V} \sqsubseteq \mathrm{ans}(Q^\Sigma|_{\tau_V}) \equiv_\Sigma \mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V}),$$

we have $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V}) \equiv_\Sigma Q$ where $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})$ is executable.

(2) By (1) and Theorem 9 we have that if $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})^\Sigma$ is defined, then there is an executable Σ_c-rewriting of Q using $\mathcal{V}$ iff $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})^\Sigma \sqsubseteq Q$. But $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})^\Sigma = \mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})^{\Sigma_b^{\mathcal{V}},\Sigma_c}$, as no chase steps with the constraints in $\Sigma_f^{\mathcal{V}}$ apply. Since Q is over τ, by Corollary 1 we have $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})^{\Sigma_b^{\mathcal{V}},\Sigma_c} \sqsubseteq Q$ iff $\mathrm{ans}(Q^{\Sigma_c,\Sigma_f^{\mathcal{V}}}|_{\tau_V})^{\Sigma_b^{\mathcal{V}},\Sigma_c}|_\tau \sqsubseteq Q$, □

Theorem 14

(1) If $\Sigma \subseteq \mathrm{IC(UCQ)}$, $\mathcal{V} \subseteq \mathrm{UCQ}$ *and there is a polynomial p such that for all* $Q, Q^{\Sigma_c} \in \mathrm{UCQ}$ *is defined and* $|Q^{\Sigma_c}| \leqslant p(|Q|)$*, then* $\textsc{Feasible}_{\Sigma,\mathcal{V}}(\mathrm{UCQ})$ *is* **NP***-complete.*
(2) If $\Sigma \subseteq \mathrm{IC}(\mathrm{UCQ}^\neg)$, $\mathcal{V} \subseteq \mathrm{UCQ}^\neg$ *and there is a polynomial p such that for all* Q, $Q^{\Sigma_c} (= \bigvee_i Q'_i) \in \mathrm{UCQ}^\neg$ *is defined and for all i:* $|Q'_i| \leqslant p(|Q|)$*, then* $\textsc{Feasible}_{\Sigma,\mathcal{V}}(\mathrm{UCQ}^\neg)$ *is* $\mathbf{\Pi_2^P}$*-complete.*

Proof. The proof is similar to that of Theorem 12. □

Corollary 7 *For any fixed* $\Sigma \subseteq \mathrm{IC}(\mathrm{UCQ}^\neg)$ *with stratified witnesses and fixed* $\mathcal{V} \subseteq \mathrm{UCQ}^\neg$, $\mathrm{FEASIBLE}_{\Sigma,\mathcal{V}}(Q^\Sigma)$ *is* $\mathbf{\Pi_2^P}$*-complete in the size of Q.*

6 Maximally Contained Rewritings

When an exact rewriting of a query Q does not exist, we want to approximate Q as best as possible. In Sections 4 and 5 we have shown how to obtain the minimally containing rewritings, which are the best overestimates of Q. In this section we consider maximally contained rewritings of Q, which are the best underestimates of Q.

Given a schema τ, let D_τ be the unary recursive query given by rules of the

form $D_\tau(x_j) :- D_\tau(x_{i_1}), \ldots, D_\tau(x_{i_k}), R(\bar{x})$ for every relation $R \in \tau$ and every access pattern R^α where $x_{i_1}, \ldots, x_{i_k}$ are the input slots of R^α and j is an output slot in R^α. Notice that every all-output access pattern yields a non-recursive rule. If no such access pattern exists, and there are no constants in the schema, then D_τ is empty on every instance and the maximally contained rewriting of Q is the empty query.

Definition 15 [Domain Extension] The *domain extension* of $Q \in \mathrm{CQ}^\neg$ is another query $\mathrm{dext}(Q)$ given by the rules with head $D_\tau(x_j)$ mentioned above and the rule

$$\mathrm{dext}(Q)(\bar{x}) :- D_\tau(y_1), \ldots, D_\tau(y_k), \mathrm{body}(Q)$$

where the head of Q is $Q(\bar{x})$ and y_i are the variables in $\mathrm{body}(Q)$.

For $Q \in \mathrm{UCQ}^\neg$ where $Q = \bigvee_i Q_i$ with $Q_i \in \mathrm{CQ}^\neg$ we define $\mathrm{dext}(Q) := \bigvee_i \mathrm{dext}(Q_i)$.

Example 6 Consider the query Q from Example 1. For this query, $\mathrm{dext}(Q)$ is the following query.

$$\begin{aligned}
D_\tau(t) &:- D_\tau(a), C(a,t)\\
D_\tau(t) &:- D_\tau(a), J(a,t)\\
D_\tau(a) &:- M(a,t)\\
D_\tau(t) &:- M(a,t)\\
D_\tau(t) &:- D_\tau(a), P(a,t,p)\\
D_\tau(p) &:- D_\tau(a), P(a,t,p)\\
D_\tau(a) &:- R(a,t)\\
D_\tau(t) &:- R(a,t)\\
D_\tau(a) &:- D(a,t,c)\\
D_\tau(t) &:- D(a,t,c)\\
D_\tau(c) &:- D(a,t,c)\\
\mathrm{dext}(Q)(a,t) &:- D_\tau(a), D_\tau(t), C(a,t)\\
\mathrm{dext}(Q)(a,t) &:- D_\tau(a), D_\tau(t), J(a,t)\\
\mathrm{dext}(Q)(a,t) &:- D_\tau(a), D_\tau(t), D_\tau(p), M(a,t), \neg P(a,t,p), L(p)
\end{aligned}$$

Notice that D_τ and $\mathrm{dext}(Q)$ are recursive queries; in particular, here we deviate from the convention in Section 2 that all the rules of a query have the same head. Clearly, $\mathrm{dext}(Q)$ is executable[7] (if all access patterns are input-only, then $\mathrm{dext}(Q)$ is equivalent to the empty query) and $\mathrm{dext}(Q) \sqsubseteq Q$. D_τ, $\mathrm{dext}(Q)$, and the following result are given in [7] for CQ.

[7] We have not defined "executable" for recursive queries, but the extension is straightforward.

Theorem 15 *If $E \sqsubseteq Q$, E is executable, and E contains no constants, then $E \sqsubseteq \text{dext}(Q) \sqsubseteq Q$.*

Proof. We already know that $\text{dext}(Q) \sqsubseteq Q$. Assume $\text{comp}(E) = \bigvee_i E_i$ with $E_1, \ldots, E_n \in \text{CQ}^\neg$. Also assume $Q := \bigvee_j Q_j$ with $Q_1, \ldots, Q_m \in \text{CQ}^\neg$. Assume $E \sqsubseteq Q$. Then $\text{comp}(E) \sqsubseteq Q$ and therefore $\forall i \exists j (Q_j \hookrightarrow E_i)$. We show that $\forall i \exists j (E_i \sqsubseteq \text{dext}(Q_j))$ and and this implies $E \equiv \text{comp}(E) \sqsubseteq \text{dext}(Q)$ as desired.

We proceed as follows. Pick i. Then there is j and a homomorphism h such that $h : Q_j \hookrightarrow E_i$. We show that $E_i \sqsubseteq \text{dext}(Q_j)$ by generating $Q' \sqsubseteq \text{dext}(Q_j)$ and an extension h' of h such that $h' : Q' \hookrightarrow E_i$. Then $E_i \sqsubseteq Q' \sqsubseteq \text{dext}(Q_j)$. To obtain Q' from Q_j we proceed as follows. Set $h^0 := h$ and $Q^0 := \text{dext}(Q_j) = D_\tau(y_1), \ldots, D_\tau(y_n), Q_j$ where $y_1, \ldots, y_n$ are the variables which appear in Q_j. If Q^m does not contain a D_τ atom, set $Q' := Q^m$ and $h' := h^m$. Otherwise, obtain Q^{m+1} from Q^m by replacing one $D_\tau(x_i)$ atom in Q^m with the body of a rule in the D_τ program containing the atom $R(\bar{x})$ in which x_i first occurs positively in an output slot in E_i (we say that the atom $R(\bar{a})$ *corresponds* to the atom $D_\tau(x_i)$). Such body may contain several D_τ atoms (those corresponding to input slots in R), but since E_i is executable, every variable x in it occurs first positively in an output slot and therefore these additional D_τ atoms correspond to atoms in E_i which occur before $R(\bar{x})$ when reading E_i from left to right. Therefore, there will be some Q^m containing no D_τ atoms. Obtain h^{m+1} by extending h^m to include the atom $R(\bar{x})$. Clearly, $Q' \sqsubseteq \text{dext}(Q_j)$ as desired. □

We must disallow constants since they can be used to partially enumerate the domain. If we allow constants and '=', we can add rules of the form $D_\tau(x) :- (x = c)$ for every constant c. Notice that nothing special needs to be done here to handle negation since negative literals do not contribute towards enumerating the domain.

Theorem 16 *If $E \sqsubseteq_\Sigma Q$, E is executable, E contains no constants, and E^Σ is defined, then $E \sqsubseteq_\Sigma \text{dext}(Q) \sqsubseteq Q$.*

Proof. If E is executable, then E^Σ is also executable. Since $E^\Sigma \sqsubseteq Q$ and contains no constants, $E^\Sigma \sqsubseteq \text{dext}(Q)$ by Theorem 15. Therefore $E \sqsubseteq_\Sigma \text{dext}(Q) \sqsubseteq Q$. □

Now assume that as in Section 5 we have a query Q, a set of constraints Σ_c, and a set of views $\mathcal{V}$ given by UCQ$^\neg$ queries $V_1, \ldots, V_n$ with access patterns on the heads of the views. We express the views as constraints $\Sigma_\text{f}^\mathcal{V}$ and $\Sigma_\text{b}^\mathcal{V}$ as in Section 5. We are interested in finding a maximally Σ_c-contained executable rewriting of Q in terms of $V_1, \ldots, V_n$. That is, we want a query over $\mathcal{V}$ that is maximally Σ-contained in Q.

Theorem 17 *If E is a maximally Σ_c-contained rewriting of Q over $\mathcal{V}$ (re-*

gardless of access patterns), and Σ_c *is such that* E^{Σ_c} *is defined for any* E*, then* $\text{dext}(E)$ *is a maximally* Σ_c*-contained executable* Σ_c*-rewriting of* Q*.*

Proof. Assume E is as in premise and P is an executable query over $\mathcal{V}$ and $P \sqsubseteq_{\Sigma_c} Q$. Then $P \sqsubseteq_{\Sigma_c} E$ by the maximality of E. Since P is executable and P^{Σ_c} is defined, by Theorem 16 we have $P \sqsubseteq_{\Sigma_c} \text{dext}(E)$. □

[7] shows how to compute such a maximally Σ_c-contained rewriting of Q in the absence of negation using a recursive plan. But it is easy to see that such recursive plans can be transformed into a union of conjunctive queries: we simply take the union of all minimal CQ queries over $\mathcal{V}$ which are Σ_c-contained in Q (the results of [7] imply that this union is finite when the chase terminates). The extension to handle negation is straightforward and we omit it in view of our results in the next section.

It turns out that in case $Q \in \text{UCQ}$ we can obtain the answer to $\text{dext}(Q)$ by using the standard certain answers semantics [8] used in information integration systems which are defined as follows [8]. The set of certain answers to Q for D under constraints Σ, which we write $\text{cert}^Q_\Sigma(D)$ is $\bigcap_{(D,T)\in\Sigma} Q(T)$. The schemas of D and T are disjoint and Σ is a set of constraints over the union of these two schemas.

Notice that the domain enumeration program D_τ is a (recursive) view and can therefore be captured with integrity constraints, as shown in Section 5. Call the set of resulting constraints Σ_D; notice that $\Sigma_D \subseteq \text{IC(CQ)}$. For every relation R and access pattern α, define a constraint $\sigma_{R,\alpha}$ as follows:

$$D_\tau(x_{i_1}), \ldots, D_\tau(x_{i_k}), R(\bar{x}) \rightarrow R'(\bar{x})$$

where $x_{i_1}, \ldots, x_{i_k}$ are the input slots of R^α. Alternatively, $D_\tau(\bar{x}), R(\bar{x}) \rightarrow R'(\bar{x})$ would work just as well.

Given a schema τ and a set of access patterns $\mathcal{P}$, define

$$\Sigma_{\tau,\mathcal{P}} := \Sigma_D \cup \{\sigma_{R,\alpha} : R \in \tau,\, \alpha \in \mathcal{P}\}.$$

$\Sigma_{\tau,\mathcal{P}}$ is a set of IC(CQ) constrains. These constraints are over the input schema $\{D_\tau\} \cup \{R : R \in \tau\}$ and output schema $\{R' : R \in \tau\}$. $\Sigma_{\tau,\mathcal{P}}$ is not source-to-target since the symbol D_τ appears both in the premise and the conclusion of some constraints in it.

Theorem 18 *For any schema* τ*, set of access patterns* $\mathcal{P}$*, query* $Q \in \text{UCQ}$*, and database* D*,* $\text{dext}(Q)(D)$ *is equal to the certain answers to* Q' *for* D *under the constraints* $\Sigma_{\tau,\mathcal{P}}$*, where* Q' *is obtained from* Q *by replacing every occurrence*

[8] Currently defined only for monotone queries.

of every relational symbol $R \in \tau$ with the corresponding symbol R'. That is,

$$\text{dext}(Q)(D) = \text{cert}^{Q'}_{\Sigma_{\tau,\mathcal{P}}}(D).$$

Therefore, access patterns do not require special treatment in information integration system beyond the introduction of additional constraints.

Proof. Assume $Q = \bigvee_i Q_i$ with each $Q_i \in \text{CQ}$. Assume $\bar{a} \in \text{dext}(Q)(D)$ and is of arity k Then $\bar{a} \in \text{dext}(Q_i)(D)$ for some i. By the definition of dext, this means that every element of $\bar{a}$ as well as every existentially-quantified witnesses for $\bar{a}$ is in D_τ. Notice that any database T satisfying $(D,T) \models \Sigma_{\tau,\mathcal{P}}$ must satisfy $R^D \cap D^r_\tau \subseteq R'^T$ for every relation $R \in \tau$ and there is such a database T_0 with $R^D \cap D^r_\tau = R'^{T_0}$. By monotonicity of Q, we must have $Q(T_0) \subseteq Q(T)$ for all databases T satisfying $(D,T) \models \Sigma\tau, \mathcal{P}$. Therefore $\bar{a} \in Q(T_0) = \text{cert}^{Q'}_{\Sigma_{\tau,\mathcal{P}}}(D)$. Conversely, assume $\bar{a} \in \text{cert}^{Q'}_{\Sigma_{\tau,\mathcal{P}}}(D)$. This implies $\bar{a} \in Q(T_0)$ and, by the definition of dext and T_0, $\bar{a} \in \text{dext}(Q)(D)$. □

7 Reducing Access Patterns to Constraints

In this section, we show that the problem $\{Q, \mathcal{P}, \Sigma\}$ of deciding feasibility in the presence of access patterns reduces to the problem $\{Q, \Sigma'\}$ of deciding equivalence in the presence of constraints only (Theorem 19). Furthermore, we reduce the problem $\{Q, \mathcal{V}, \mathcal{P}, \Sigma\}$ of finding rewritings using views with access patterns to one of finding rewritings using views and constraints in the absence of access patterns $\{Q, \mathcal{V}, \Sigma''\}$ (Theorem 20). These results enable alternative proofs for the complexity of answering queries in the presence of access patterns. They also facilitate an alternative implementation of algorithms REWRITE, FEASIBLE, VIEWREWRITE etc. using a chase-based module for rewriting under constraints such as the **C&B** implementation in [4]. The reduction uses the constraints Σ_D defined in the previous section.

Theorem 19 *If Σ has stratified witnesses, then*

(1) Q is Σ-feasible iff $Q \sqsubseteq_{\Sigma_D \cup \Sigma} \text{dext}(Q)$, and
(2) $Q^{\Sigma_D \cup \Sigma}$ is defined and $Q \sqsubseteq_{\Sigma_D \cup \Sigma} \text{dext}(Q)$ is decidable in $\mathbf{\Pi^P_2}$ in the size of Q.

Proof. Part (1): The proof uses Lemmas 10 and 11 below.

Only If: Assume Q is Σ-feasible. Then there exists an executable E such that $Q \equiv_\Sigma E$. In particular, $E \sqsubseteq_\Sigma Q$. Moreover, since Σ has stratified witnesses, E^Σ is defined. Then by Theorem 16, $E \sqsubseteq_\Sigma \text{dext}(Q)$. Since $Q \sqsubseteq_\Sigma E$, we have $Q \sqsubseteq_\Sigma \text{dext}(Q)$.

If: Assume $Q \sqsubseteq_{\Sigma_D \cup \Sigma} \text{dext}(Q)$. By construction, $\text{dext}(Q) \sqsubseteq Q$ so $\text{dext}(Q) \equiv_{\Sigma_D \cup \Sigma} Q$. Since $\text{dext}(Q)$ is executable, Q is $\Sigma_D \cup \Sigma$-feasible. Then by Corollary 4, we have that $\text{ans}(Q^{\Sigma_D \cup \Sigma})^{\Sigma_D \cup \Sigma} \sqsubseteq Q$. By Lemma 11 below, $\text{ans}(Q^{\Sigma_D \cup \Sigma})^{\Sigma} \sqsubseteq Q$.

But from Lemma 10 below it follows that $\text{ans}(Q^{\Sigma_D \cup \Sigma})^{\Sigma}$ is the same as $\text{ans}(Q^{\Sigma})^{\Sigma}$ enriched with D_τ atoms, because D_τ atoms do not appear in Σ and thus do not contribute to the second chase. Moreover, since D_τ does not appear in Q, the homomorphisms witnessing the containment $\text{ans}(Q^{\Sigma_D \cup \Sigma})^{\Sigma} \sqsubseteq Q$ also witness the containment $\text{ans}(Q^{\Sigma})^{\Sigma} \sqsubseteq Q$. Therefore, $\text{ans}(Q^{\Sigma}) \sqsubseteq_{\Sigma} Q$, whence $\text{ans}(Q^{\Sigma}) \equiv_{\Sigma} Q$. Since $\text{ans}(Q^{\Sigma})$ is executable, Q is Σ-feasible.

Part (2): The termination of the chase is guaranteed by the combination of two reasons. First, Σ has stratified witnesses, so Q^{Σ} is defined. Second, though Σ_D does *not* have stratified witnesses, we can show that only the "forward" constraints from Σ_D apply during the chase of Q. These contain no disjunction and no existential quantification. Moreover, they only introduce unary D-atoms, so they only increase the result of chasing Q with Σ alone by a linear factor. □

Lemma 10 *Set $Q' = Q^{\Sigma, \Sigma_D}$. Chasing Q^{Σ} with Σ_D introduces an atom $D_\tau(x)$ iff x appears in some Q'-answerable literal of Q'. Equivalently, a literal $l(x_1, \ldots, x_n)$ in Q' is Q'-answerable iff the chase of Q^{Σ} with Σ_D introduces $D_\tau(x_1), \ldots, D_\tau(x_n)$.*

Proof. This follows immediately from the definition of Σ_D and answerable literals. □

Lemma 11 $\text{ans}(Q^{\Sigma_D \cup \Sigma})^{\Sigma_D \cup \Sigma} = \text{ans}(Q^{\Sigma, \Sigma_D})^{\Sigma, \Sigma_D} = \text{ans}(Q^{\Sigma, \Sigma_D})^{\Sigma}$.

Proof. The first equality holds since Σ does not mention D_τ. For the second equality, set $Q' = Q^{\Sigma, \Sigma_D}$. Notice that chasing $\text{ans}(Q')$ with Σ does not introduce Q'-answerable literals because any such literals are already in $\text{ans}(Q')$. Therefore, by Lemma 10, the chase of $\text{ans}(Q')^{\Sigma}$ with Σ_D does not apply. □

Theorem 19 gives another route to the upper bound on the complexity of checking feasibility from the complexity of checking containment.

We now reduce the rewriting problem $\{Q, \mathcal{V}, \mathcal{P}, \Sigma\}$ to $\{Q, \mathcal{V}, \Sigma''\}$. First, define D_τ as in Section 6, but using view symbols instead of relation symbols from τ. Next capture D_τ with constraints Σ_D. For any $V \in \mathcal{V}$ and access pattern V^α, define a new view $V^D(\text{free}(V)) :- \text{body}(V), D(x_{i_1}), \ldots, D(x_{i_k})$ where the $x_{i_j} \in \text{free}(V)$ are the free variables of V which appear in input slots. Each V^D is a view without access patterns. Denoting $\mathcal{V}^D := \{V^D \mid V \in \mathcal{V}\}$, we have the following result.

Theorem 20 *Q has an exact (minimally containing) executable Σ-rewriting over $\mathcal{V}$ iff it has an exact (minimally containing) $\Sigma \cup \Sigma_D$-rewriting over $\mathcal{V}^D$.*

Proof. If: Let R^D be a $\Sigma \cup \Sigma_D$-rewriting of Q over $\mathcal{V}^D$. Our candidate for the executable Σ-rewriting of Q is the recursive program P defined by the union of the following rules

- the rules in D_τ,
- for each $V^D(\bar{x}, \bar{y}) :- D(\bar{y}), \text{body}(V)$ in $\mathcal{V}^D$, the rule $V'(\bar{x}, \bar{y}) :- D(\bar{y}), V(\bar{x}, \bar{y})$, and
- the rule obtained by replacing each occurrence of V^D in R^D with V'.

Indeed, notice that $P \equiv_{\Sigma_D} R^D$ by construction. Since $R^D \equiv_{\Sigma_D \cup \Sigma} Q$, we have $P \equiv_{\Sigma \cup \Sigma_D} Q$. From the semantics of the datalog rules for D_τ, it follows that $P \equiv_\Sigma Q$. But P is executable and its EDBs are in $\mathcal{V}$.

Only If: Let Q have an executable Σ-rewriting R over $\mathcal{V}$, i.e. $Q \equiv_{\Sigma_\text{f}^\mathcal{V} \cup \Sigma_\text{b}^\mathcal{V} \cup \Sigma} R$. Since R is executable, R is feasible over $\mathcal{V}$, in the presence of $\Sigma_\text{f}^\mathcal{V} \cup \Sigma_\text{b}^\mathcal{V} \cup \Sigma$, so by Theorem 19, $R \equiv_{\Sigma_\text{f}^\mathcal{V} \cup \Sigma_\text{b}^\mathcal{V} \cup \Sigma \cup \Sigma_D} \text{dext}(R)$. Then $Q \equiv_{\Sigma_\text{f}^\mathcal{V} \cup \Sigma_\text{b}^\mathcal{V} \cup \Sigma \cup \Sigma_D} \text{dext}(R)$. Notice that by construction of $\text{dext}(R)$, $\text{dext}(R)$ is equivalent to a query R^D obtained by replacing each occurrence of $V \in \mathcal{V}$ in R with V^D. Then R^D is a $\Sigma \cup \Sigma_D$-rewriting of Q over $\mathcal{V}^D$. $\square$

Theorem 20 enables an alternative implementation of algorithm ViewRewrite, namely by rewriting using views and integrity constraints in the absence of access patterns. [4] describes the implementation of the **C&B** algorithm, which is sound and complete for precisely this rewriting task (i.e. it finds a $\{Q, \mathcal{V}, \Sigma\}$-rewriting whenever one exists). All we need to do to use the **C&B** implementation is to apply it to $\mathcal{V}^D$ and $\Sigma \cup \Sigma_D$ instead of $\mathcal{V}$ and Σ.

Remark. The following observation sheds additional light on why the rewriting problem $\{Q, \mathcal{V}, \mathcal{P}, \Sigma\}$ reduces to $\{Q, \mathcal{V}, \Sigma''\}$. We can show that the answerable part $\text{ans}(Q)$ of a query Q can be computed by chasing Q with Σ_D (as obtained for Theorem 19 and Theorem 20). More specifically,

(a) $\text{chase}(Q, \Sigma_D, O)$ terminates for any order O on Σ_D,
(b) the chase result is unique regardless of O (denote it Q^{Σ_D}), and
(c) if we restrict Q^{Σ_D} to only those atoms $R(\bar{x})$ for which $D(\bar{x})$ appears in Q^{Σ_D}, we obtain $\text{ans}(Q)$.

8 Extensions

The key technique that allows us to treat negation, views, and access patterns uniformly is modeling with constraints (recall $\Sigma_\neg^\tau$, $\Sigma_\text{f}^\mathcal{V} \cup \Sigma_\text{b}^\mathcal{V}$, respectively Σ_D). This approach enables the straightforward implementation of our algorithms by reusing an already existing chase module [4]. It turns out that we can extend

our solution to handling equality and arithmetic comparisons by capturing them with constraints as well.

Handling Equality. Equality can be modeled as a binary relation E with access patterns 'io' and 'oi' subject to the following constraints $\Sigma^{\tau}_{=} \subseteq \mathrm{IC(CQ)}$:

- $\forall x, y \;\; E(x, y) \rightarrow E(y, x)$,
- $\forall x, y, z \;\; E(x, y), E(y, z) \rightarrow E(x, z)$,
- for every $R \in \tau$ and i, $\forall \bar{x} \;\; R(\bar{x}) \rightarrow E(x_i, x_i)$, and
- for every $R \in \tau$: $\forall \bar{x}, \bar{y} \;\; R(\bar{x}) \wedge E(x_1, y_1) \wedge \ldots \wedge E(x_k, y_k) \rightarrow R(\bar{y})$.

Handling Arithmetic Comparisons. The comparison '$\leq$', which gives $\mathrm{UCQAC}^{\neg}$, can be handled as a binary relation LE with access pattern 'ii' subject to the following constraints $\Sigma_{\leq} \subseteq \mathrm{IC(CQ}^{\neg})$ which say that LE is an unbounded dense total ordering:

(1) $\forall x, y, z \;\; LE(x, y) \wedge LE(y, z) \rightarrow LE(x, z)$,
(2) $\forall x, y \;\; LE(x, y) \wedge LE(y, x) \rightarrow E(x, y)$,
(3) $\forall x, y \;\; \neg LE(x, y) \rightarrow LE(y, x)$, and
(4) $\forall x, y \;\; L(x, y) \rightarrow \exists u, v, w \; (L(u, x) \wedge L(x, v) \wedge L(v, y) \wedge L(y, w))$,

where $L(x, y)$ stands for $LE(x, y) \wedge \neg E(x, y)$.

Notice that the chase with axiom (4) (the density axiom) is non-terminating, yielding chains of $<$ comparisons of arbitrary length. However, we can show that for a natural restriction, there is no need to chase with the density axiom. This restriction demands that all integrity constraints be safe, i.e. that all variables appearing in a $\leq$ atom also appear in some relational atom other than a $\leq$ atom. In this case, all of our results extend to unions of conjunctive queries with negation, equality and arithmetic comparisons ($\mathrm{UCQAC}^{\neg}$) as well as the corresponding constraints $\mathrm{IC(UCQAC}^{\neg})$. All we need to do is replace Σ with Σ' consisting of Σ, the equality constraints, and constraints 1-3 above, then run algorithms Feasible, ViewRewrite, Rewrite, ViewFeasible on Σ'.

Even if the restriction above does not hold, it can be shown that the chase with the density axiom can be truncated so as to generate $<$ chains of length bounded by the number of variables in the original query. All we need to do is run algorithms Feasible, ViewRewrite, Rewrite, ViewFeasible using the truncating chase with the constraints $\Sigma'' := \Sigma \cup \Sigma^{\tau}_{=} \cup \Sigma_{\leq}$.

Queries with Binding Patterns.

So far, we have not considered binding patterns in the query to be answered.

This is the same as considering queries with an all-output annotation on the query head. But suppose we want to answer a query Q with an annotation Q^{io}. That is, we are willing to supply a value in the first slot of Q in order to get an answer to our query. We can model such queries in two ways. (1) We can replace the free variables in Q with 'i' annotations with new constants to obtain Q'. (2) We can add a new unary relation S and a corresponding unary view (in case we are working with views) and give this unary relation/view the access pattern 'o'. Then we can obtain the query Q' from Q by adding $S(x)$ to the query body for every variable x in the head with an 'i' access pattern. Either way, we have that Q is feasible, Σ-feasible, etc. for such access patterns in Q iff Q' is feasible, Σ-feasible, etc.

9 Answerability

Feasibility is defined in terms of $\text{UCQ}^\neg$ queries. A natural question is whether some $\text{UCQ}^\neg$ queries over sources with binding patterns may be answered not by a $\text{UCQ}^\neg$ query, but by a more general query. In the most general case, we may allow any computable query. However, to model the presence of binding patterns, we must define such a computable query not in terms of a Turing machine which takes the relation extents (which may be inaccessible) as inputs, but instead as a Turing machine with oracle access to the relations, subject to the access patterns. We say that an oracle call to R is *compatible* with the access patterns α if the input to the oracle call is a set of tuples I of the same arity as the number of input slots in R^α and the output is the set of tuples O in R such that O restricted to the input slots in R^α is equal I.

Definition 16 We say that Q is answerable if there is an oracle Turing machine M taking no input which, for any database D, can make queries to the base relations of D which are *compatible* with the access patterns and such that M computes $Q(D)$.

Theorem 21 *Q is feasible iff it is answerable.*

Proof. Fix some schema τ and assume Q is over τ. It is clear that if Q is feasible, then it is answerable, so we prove the converse. Assume $Q = \bigvee_i Q_i$ with each $Q_i \in \text{CQ}^\neg$.

Assume first that $\text{ans}(Q)$ is defined, but $\text{ans}(Q) \not\equiv Q$. By Lemma 8 we must have $\text{ans}(Q) \not\sqsubseteq Q$. Therefore, there is k such that $\text{ans}(Q_k) \not\sqsubseteq Q$. Assume that $\bar{x}$ are the free variables of Q_k. Set $A := [Q_k]$ and $B := [\text{ans}(Q_k)]$. Then $c_{\bar{x}} \in Q(A) - Q(B)$ so $Q(A) \neq Q(B)$. For each relation R in τ, set $R' := \text{dext}(R)$. Then $R'(A) = R'(B)$ by Lemma 12 below. Therefore, all oracle calls on databases A and B give equal outputs on equal inputs. This implies that

any oracle Turing machine which only makes queries with tuples over D_τ must give the same answer for databases A and B, and therefore can not compute Q. In fact, this holds for an arbitrary oracle Turing machine with no input which is only allowed to make queries to the relations R in τ, since such machine can only "invent" a finite number of additional constants C not in D_τ and we can take isomorphic copies of A and B whose universe is disjoint from C.

If $\text{ans}(Q)$ is undefined, then for some k $\text{ans}(Q_k)$ has fewer free variables than Q_k and therefore also $Q_k \not\sqsubseteq Q$, so the argument above also applies. □

The following lemma follows directly from the definitions of ans and dext.

Lemma 12 *If* $R' = \text{dext}(R)$ *and* $Q \in \text{CQ}^\neg$*, then* $R'([Q]) = R'([\text{ans}(Q)])$.

10 Conclusions

Our results extend previous work in two directions. (1) We treat access patterns, constraints, and views together. (2) We allow for larger classes of queries, constraints, and views that include negation and arithmetic comparisons. We present a framework for these results that unifies several techniques presented separately in previous work. We also show how to handle access patterns in an information integration setting. Finally, we show that it is sufficient to search for executable rewritings only among $\text{UCQ}^\neg$ queries.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *ACM Symposium on Theory of Computing (STOC)*, pages 77–90, 1977.

[3] Alin Deutsch, Bertram Ludäscher, and Alan Nash. Rewriting queries using views with access patterns under integrity constraints. In *Intl. Conference on Database Theory (ICDT)*, 2005.

[4] Alin Deutsch and Val Tannen. Mars: A system for publishing xml from mixed and redundant storage. In *Intl. Conf. on Very Large Data Bases (VLDB)*, 2003.

[5] Alin Deutsch and Val Tannen. Reformulation of xml queries and constraints. In *Intl. Conf. on Database Theory (ICDT)*, 2003.

[6] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *PODS*, 1997.

[7] Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. Recursive query plans for data integration. *Journal of Logic Programming*, 43(1):49–73, 2000.

[8] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Intl. Conf. on Database Theory (ICDT)*, 2003.

[9] Daniela Florescu, Alon Y. Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *SIGMOD*, pages 311–322, 1999.

[10] J. Grant and J. Minker. A logic-based approach to data integration. *Theory and Practice of Logic Programming*, 2(3):323–368, 2002.

[11] Alon Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[12] Richard Hull and Masatoshi Yoshikawa. On the equivalence of database restructurings involving object identifiers. In *PODS*, 1991.

[13] Christoph Koch. Query rewriting with symmetric constraints. *AI Communications*, 17(2), 2004. to appear.

[14] Alon Y. Levy. Logic-based techniques in data integration. In Jack Minker, editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC*, 1999.

[15] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *22nd Intl. Conf. on Very Large Data Bases (VLDB)*, pages 251–262, Bombay, India, 1996.

[16] Chen Li. Computing complete answers to queries in the presence of limited access patterns. *Journal of VLDB*, 12:211–227, 2003.

[17] Chen Li and Edward Y. Chang. On answering queries in the presence of limited access patterns. In *Intl. Conference on Database Theory (ICDT)*, 2001.

[18] Todd D. Millstein, Alon Y. Levy, and Marc Friedman. Query containment for data integration systems. In *PODS*, pages 67–75, 2000.

[19] Alan Nash and Bertram Ludäscher. Processing first-order queries under limited access patterns. In *PODS*, Paris, France, 2004.

[20] Alan Nash and Bertram Ludäscher. Processing unions of conjunctive queries with negation under limited access patterns. In *Intl. Conference on Extending Database Technology (EDBT)*, Heraklion, Crete, Greece, 2004.

[21] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *PODS*, pages 105–112, 1995.

[22] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1980.

[23] Jeffrey Ullman. The complexity of ordering subgoals. In *PODS*, 1988.

[24] Vasilis Vassalos and Yannis Papakonstantinou. Expressive capabilities description languages and query rewriting algorithms. *Journal of Logic Programming*, 43(1):75–122, 2000.

A Some Proofs

Proof. (**Lemma 7**) Assume $Q = \bigvee_k Q_k$ with $Q_1, \ldots, Q_r \in \mathrm{CQ}^\neg$. Assume $\mathrm{ans}(Q)$ is defined. Then, by Lemma 6, $\mathrm{ans}(\mathrm{comp}(Q))$ is defined since $\mathrm{comp}(Q)$ is complete and $\mathrm{comp}(Q) \sqsubseteq \mathrm{ans}(Q)$. Set $A = \mathrm{ans}(\mathrm{comp}(Q))$ and $E = \mathrm{comp}(\mathrm{ans}(Q))$. Assume $A = \bigvee_i A_i$ and $E = \bigvee_j E_j$ with $A_1, \ldots, A_n, E_1, \ldots, E_m \in \mathrm{CQ}^\neg$. Also assume $Q' = \mathrm{comp}(Q) = \bigvee_\ell Q'_\ell$ with $Q_1, \ldots, Q_s \in \mathrm{CQ}^\neg$. Notice that all A_i, E_j, and Q'_ℓ are satisfiable, by the definition of comp.

Pick an arbitrary A_i. Then $A_i = \mathrm{ans}(Q'_\ell)$ for some ℓ with Q'_ℓ complete and there must be some k such that $Q'_\ell \sqsubseteq Q_k$. By Theorem 2, $Q_k \hookrightarrow Q'_\ell$ and therefore $\mathrm{ans}(Q_k) \hookrightarrow Q'_\ell$. Again by Theorem 2, $Q'_\ell \sqsubseteq \mathrm{ans}(Q_k)$ and by Lemma 6, $\mathrm{ans}(Q'_\ell) \sqsubseteq \mathrm{ans}(Q_k)$ which yields $A_i = \mathrm{ans}(Q'_\ell) \sqsubseteq \mathrm{ans}(Q_k) \sqsubseteq \mathrm{ans}(Q)$. Since this holds for all i, by Theorem 3, $\mathrm{ans}(\mathrm{comp}(Q)) = A \sqsubseteq \mathrm{ans}(Q)$.

Pick an arbitrary E_j. Then $E_j \sqsubseteq \mathrm{ans}(Q_k)$ for some k. By Theorem 2, there is a homomorphism $h : \mathrm{ans}(Q_k) \to E_j$. We show below that, for some ℓ, we can extend h to a homomorphism $h' : \mathrm{ans}(Q'_\ell) \to E_j$. Therefore, by Theorem 2, $E_j \sqsubseteq \mathrm{ans}(Q'_\ell) = A_i$ for some i. Since this holds for all i, by Theorem 3 and Lemma 4, $\mathrm{ans}(Q) \equiv E \sqsubseteq A = \mathrm{ans}(\mathrm{comp}(Q))$.

To extend h we proceed as follows. To get from Q_k to $Q'_\ell \sqsubseteq Q_k$ we need to add literals to Q_k. We set $P_0 = Q_k$ and $P_t = Q_\ell$ and we add only one literal to each P_i to get P_{i+1}.[9] Similarly, we will have homomorphisms $h_i : \mathrm{ans}(P_i) \to E_j$ starting with $h_0 = h$ and ending with $h_t = h'$.

To go from P_i to P_{i+1} we have two possibilities: either we add $r(\bar{x})$ or we add $\neg r(\bar{x})$ where $r(\bar{x})$ does not occur in P_i. Suppose that $\neg r(\bar{x})$ is not P_i-answerable. Then we set $P_{i+1} = P_i, \neg r(\bar{x})$ and $h_{i+1} = h_i$. Since $\mathrm{ans}(P_{i+1}) = \mathrm{ans}(P_i)$, we have the desired homomorphism $h_{i+1} : \mathrm{ans}(P_{i+1}) \to E_j$.

Otherwise, since E_j is complete, there must be a homomorphism $h_{i+1} : \mathrm{ans}(P_i), r(\bar{x}) \to E_j$ or a homomorphism $h_{i+1} : \mathrm{ans}(P_i), \neg r(\bar{x}) \to E_j$. We set P_{i+1} accordingly to either $P_i, r(\bar{x})$ or $P_i, \neg r(\bar{x})$. In the former case, since $\neg r(\bar{x})$ is not P_i-answerable, $r(\bar{x})$ does not add any new bindings and therefore $\mathrm{ans}(P_i, r(\bar{x})) = \mathrm{ans}(P_i), r(\bar{x})$ and we have the desired homomorphism $h_{i+1} : \mathrm{ans}(P_{i+1}) \to E_j$. □

Proof. (**Lemma 6**) Assume $Q, E \in \mathrm{CQ}^\neg$, Q is complete, and $Q \sqsubseteq \mathrm{ans}(E)$. Then, by Theorem 2, there is a homomorphism $h : \mathrm{ans}(E) \to Q$. Define $\mathrm{body}(Q')$ as the conjunction of all the literals in the range of h (possibly with repetition) in the order induced by $\mathrm{ans}(E)$. We know that in $\mathrm{body}(\mathrm{ans}(E))$, every variable appears first in an output slot of a positive literal. Since h maps positive literals to positive literals, the same holds for $\mathrm{body}(Q')$. In particular,

[9] the order is unimportant

h must map every variable in $\text{head}(\text{ans}(E)) = \text{ans}(E)$ to a different variable in $\text{head}(Q)$, and therefore since $\text{ans}(E)$ is defined and therefore safe, $\text{ans}(Q')$ is defined. Furthermore, every literal in Q' is Q'-answerable and therefore Q-answerable and thus appears in $\text{ans}(Q)$, which is defined because $\text{ans}(Q')$ is defined. Therefore h is a homomorphism $h : \text{ans}(E) \to \text{ans}(Q)$ and, by Theorem 2, $\text{ans}(Q) \sqsubseteq \text{ans}(E)$. □

Proof. (**Lemma 8**) Set $Q' := \text{comp}(Q)$ and assume $Q' := \bigvee_i Q_i$ with $Q_i \in \text{CQ}^\neg$. Since the identity mapping is a homomorphism $\text{ans}(Q_i) \to Q_i$, by Theorem 3, $\text{comp}(Q) \sqsubseteq \text{ans}(\text{comp}(Q))$. By Lemmas 4 and 7,

$$Q \equiv \text{comp}(Q) \sqsubseteq \text{ans}(\text{comp}(Q)) \equiv \text{ans}(Q).$$

□

Proof. (**Theorem 5**) Assume $Q \sqsubseteq E$ and E is executable. Since, by Lemma 4, $\text{comp}(Q) \equiv Q$, we have $\text{comp}(Q) \sqsubseteq E$. Assume $\text{comp}(Q) = \bigvee_i Q_i$ and $E = \bigvee_j E_j$ with $Q_i, E_j \in \text{CQ}^\neg$. Then since $\text{comp}(Q)$ is complete, by Theorem 3, $\forall i \exists j (Q_i \sqsubseteq E_j)$. Since $\text{ans}(E_j) = E_j$, by Lemma 6, $\text{ans}(Q)$ is defined and $\forall i \exists j (\text{ans}(Q_i) \sqsubseteq E_j)$. By Lemmas 8, 7, and Theorem 3,

$$Q \sqsubseteq \text{ans}(Q) \equiv \text{ans}(\text{comp}(Q)) \sqsubseteq E.$$

□

Proof. (**Theorem 6**) Corollary 2 shows that $\text{FEASIBLE}(\text{UCQ}^\neg) \leqslant_m^{\text{P}} \text{CONT}(\text{UCQ}^\neg)$ (if $\text{ans}(Q)$ is undefined, we reduce Q to two queries P, P' such that $P \not\sqsubseteq P'$).

For the other direction, consider two queries $P, Q \in \text{UCQ}^\neg$ with free variables $\bar{x}$ where $P = P_1 \vee \ldots \vee P_k$. We define the query

$$P'(\bar{x}, y) :- P_1, B(y) \vee \ldots \vee P_k, B(y)$$

where y is a variable not appearing in P or Q and B is a relation not appearing in P or Q with access pattern B^{i}. We give relations R appearing in P or Q output access patterns (i.e., $R^{\text{ooo}\cdots}$). As a result, P and Q are both executable, but $P' \sqsubset P$ and P' is not feasible. We set $Q' := P' \vee Q$. Clearly, $\text{ans}(Q') \equiv P \vee Q$. If $P \sqsubseteq Q$, then $\text{ans}(Q') \equiv P \vee Q \equiv Q \sqsubseteq Q'$ so by Corollary 2, Q' is feasible. If $P \not\sqsubseteq Q$, then since $P' \sqsubset P$ and $P' \not\sqsubseteq Q$ we have $\text{ans}(Q') \equiv P \vee Q \not\sqsubseteq P' \vee Q \equiv Q'$ so again by Corollary 2, Q' is not feasible. This shows that $\text{CONT}(\text{UCQ}^\neg) \leqslant_m^{\text{P}} \text{FEASIBLE}(\text{UCQ}^\neg)$. □

Proof. (**Theorem 8**) Part (1). In the following, given a quantifier-free formula φ, we denote with $\varphi[\bar{x}]$ a query with body φ and tuple of head variables $\bar{x}$. Given a mapping h, $h(\varphi)$ denotes the formula obtained by substituting $h(\text{vars}(\varphi))$ for $(\text{vars}(\varphi))$.

Assume σ is normalized, of form (11). Let $\text{step}(Q, \Sigma, h) := \bigvee_{i=1}^{n} Q_i$. By definition of the chase step, each Q_i is obtained by adding goals to Q. Therefore $Q_i \sqsubseteq Q$ for each $1 \leqslant i \leqslant n$ and $\text{step}(Q, \Sigma, h) \sqsubseteq Q$. We next show $Q \sqsubseteq_\Sigma \bigvee_i Q_i$. Pick an arbitrary database D and an arbitrary tuple $\bar{d}$ such that $D\bar{d} \models Q$ as witnessed by a mapping v. We have

$$
\begin{array}{lllll}
 & D\ \bar{d} & \models Q & \text{witness } v & (s1) \\
\Rightarrow & D\ v \circ h(\bar{x}) & \models \psi[\bar{x}] & \text{witness } v \circ h & (s2) \\
\Rightarrow \exists i & D\ v \circ h(\bar{x}) & \models \psi \wedge \xi_i[\bar{x}] & \text{witness } h_i & (s3) \\
\Rightarrow \exists i & D\ v \circ h(\bar{x}) & \models h'(\psi \wedge \xi_i)[h'(\bar{x})] & \text{witness } v' & (s4) \\
\Rightarrow \exists i & D\ \bar{d} \cap v \circ h(\bar{x}) & \models h'(\psi \wedge \xi_i)[h'(\bar{x}) \cap \text{free}(Q)] & \text{witness } v' & (s5) \\
\Rightarrow \exists i & D\ \bar{d} & \models \text{body}(Q) \wedge h'(\psi \wedge \xi_i)[\text{free}(Q)] & \text{witness } v' & (s6) \\
\Rightarrow \exists i & D\ \bar{d} & \models Q_i & \text{witness } v' & (s7)
\end{array}
$$

where

- $\bar{x} := \text{vars}(\psi)$
- h_i is a mapping on $\text{vars}(\sigma)$ which agrees with $v \circ h$ on $\bar{x}$
- h' agrees with h on $\bar{x}$ and is the identity on $\bar{y}_i$
- v' agrees with v on $\text{vars}(Q)$ and with h_i on $\bar{y}_i$
- $\bar{d} \cap v \circ h(\bar{x})$ denotes the tuple obtained by keeping only the components in $\bar{d}$ (their relative order is preserved) which appear in some component of $v \circ h(\bar{x})$. Analogously for $h'(\bar{x}) \cap \text{free}(Q)$.

(s2) follows from (s1). (s3) follows from (s2) and the fact that $D \models \sigma$. (s4) from (s3) and the observations that $v' \circ h'(\bar{x}) = v \circ h(\bar{x})$ and that $v' \circ h'(\bar{y}_i) = h_i(\bar{y}_i)$. (s5) follows trivially from (s4) and so does (s6) from (s5). (s7) follows from (s6) and the definition of the chase step.

Part (2). By induction on the length of the chase sequence, using Part (1) for the induction step. $\square$

Proof. (**Theorem 9**) We first observe that $P \sqsubseteq_\Sigma Q$ iff $P \sqsubseteq_{\Sigma^\tau_\neg \cup \Sigma} Q$ since all databases over the same schema as $\Sigma^\tau_\neg$ satisfy $\Sigma^\tau_\neg$, in particular so do all databases which satisfy Σ. It suffices therefore to prove

$$P \sqsubseteq_{\Sigma^\tau_\neg \cup \Sigma} Q \text{ iff } \text{chase}(P, \Sigma^\tau_\neg \cup \Sigma, O) \sqsubseteq Q$$

If: Observe that $\text{chase}(P, \Sigma^\tau_\neg \cup \Sigma, O) \sqsubseteq Q$ implies $\text{chase}(P, \Sigma^\tau_\neg \cup \Sigma, O) \sqsubseteq_{\Sigma^\tau_\neg \cup \Sigma} Q$ and that Theorem 8 implies $P \sqsubseteq_{\Sigma^\tau_\neg \cup \Sigma} \text{chase}(P, \Sigma^\tau_\neg \cup \Sigma, O)$.

Only If: Assume $\text{chase}(P, \Sigma^\tau_\neg \cup \Sigma, O) = \bigvee_{i=0}^{k} P_i$ with each $P_i \in \text{CQ}^\neg$. Notice

that if $k = 0$ then the chase result is defined as the unsatisfiable empty union, which is contained in any Q. Suppose $k > 0$. Let $\bar{x}$ be the free variables of P_i Since P_i is satisfiable, it follows from Lemma 1 that $[P_i] \models P_i$, which implies $[P_i] \models \text{chase}(P, \Sigma^\tau_\neg \cup \Sigma, O)$. But since the chase terminates, it follows that $[P_i] \models \Sigma^\tau_\neg \cup \Sigma$ for each $1 \leqslant i \leqslant k$ hence by soundness of the chase (Theorem 8) we have $[P_i] \models P$ and by hypothesis we conclude $[P_i] \models Q$. Assume $Q = \bigvee_{j=1}^{l} Q_j$ where $Q_j \in \text{CQ}^\neg$ for all $1 \leqslant j \leqslant l$. Then by Theorem 3 for some j, $Q_j \hookrightarrow [P_i]$. Now observe that since P_i was obtained by chasing with $\Sigma^\tau_\neg$, it is complete, hence by Lemma 3 we have $Q_j \hookrightarrow P_i$ whence by Theorem 2 we get $\text{chase}(P, \Sigma^\tau_\neg \cup \Sigma) \sqsubseteq Q$. $\square$

Proof. (**Theorem 10**) Assume that Q, Σ, e, u, and l are as in the hypotheses of the Theorem.

Part 1: We show that the chase can generate only a finite number of new variables and hence only finitely many distinct new atoms. Notice that after all these atoms are generated, no more chase steps can apply (they would only generate duplicates of already existing atoms) and the chase must terminate.

For any query Q' in the chase sequence and any $v \in \text{vars}(Q')$, define

$$\text{name}(v) = \begin{cases} v & \text{if } v \in \text{vars}(Q) \\ F_{\sigma,i,k}(\text{name}(h(\bar{z}))) & \text{if (*) holds} \end{cases}$$

where (*) is the property that v was generated during the chase prefix ending at Q', as the result of a chase step with $\sigma \in \Sigma$ using homomorphism h. Furthermore, σ has general form (11) and v is the variable corresponding to the k^{th} component of $\bar{y}_i$. We define $\bar{z} := \bar{x} \cap \text{vars}(\xi_i)$. $\{F_{\sigma,i,k}\}_{\sigma,i,k}$ is a family of Skolem function symbols.

Notice that names are terms over the variables of Q and the Skolem function symbols. The following is easily shown by induction on the length of the chase sequence.

Claim 1. *Names uniquely identify the variables generated during the chase.*

For any name n we define the *depth* of n as the maximum nesting depth of Skolem function symbols in n. Then we can prove the following by induction on d (omitted).

Claim 2. *Let v be any variable satisfying (*) above, and let* $\text{depth}(\text{name}(v)) = d$. *If v appears in ξ_i as the m^{th} argument of a relation R then there exists a path p in the chase flow graph of Σ such that p ends in the node $R.m$ and contains d $\exists$-edges.*

Since the chase flow graph G has no cycles through $\exists$-edges, the maximum

number (over all paths in G) of $\exists$-edges per path is well-defined and finite and we denote it l. By Claim 2, the depth of variable names is upper bounded by l. Since the number of distinct variables is bounded by Q and the number of distinct Skolem symbols is bounded by Σ, there are only finitely many distinct names the chase can produce. By Claim 1, this results in finitely many distinct variables.

Part 2: Recall that at every step of the chase, we can view the intermediate result as a finite set of $\mathrm{CQ}^{\neg}$ queries. Denoting with M_j the number of variable names of depth at most j appearing in one of these queries, we have the following recurrence relation

$$\begin{aligned} M_0 &\leq |\mathrm{vars}(Q)| \\ M_{j+1} &\leq M_j + eM_j^u \end{aligned}$$

since there are at most e Skolem functions and each term of depth $j+1$ consists of a Skolem function with at most u arguments, each a Skolem term of depth j. Since

$$M_{j+1} \leq M_j + eM_j^u \leq (1+e)M_j^u \leq (1+e)^u M_j^u$$

it follows that $M_l \leq (1+e)^{u^l} |\mathrm{vars}(Q)|^{u^l}$. $\square$