Semantic Query Optimization in the Presence of Types

Michael Meier, Michael Schmidt*, Fang Wei, and Georg Lausen
Institut für Informatik, University of Freiburg
Freiburg i. Br., Germany
{meierm, mschmidt, fwei, lausen}@informatik.uni-freiburg.de

ABSTRACT

Both semantic and type-based query optimization rely on the idea that queries often exhibit non-trivial rewritings if the state space of the database is restricted. Despite their close connection, these two problems to date have always been studied separately. We present a unifying, logic-based framework for query optimization in the presence of data dependencies and type information. It builds upon the classical chase algorithm and extends existing query minimization techniques to considerably larger classes of queries and dependencies. In particular, our setting requires chasing conjunctive queries (possibly with union and negation) in the presence of dependencies containing negation and disjunction. We study the applicability of the chase in this setting, develop novel conditions that guarantee its termination, identify fragments for which minimal query computation is always possible (w.r.t. a generic cost function), and investigate the complexity of related decision problems.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems - relational databases, query processing

General Terms: Algorithms, theory

Keywords: Query optimization, types, constraints, chase

1. INTRODUCTION

Typing is a central component of many practical database systems, including (but not limited to) relational databases, object-oriented database models [25, 35], typed datalog [42], and semi-structured data [33]. In response, to date a rich theory of type-based optimization has been developed [38, 19, 29, 28, 21, 23, 4]. These optimization approaches often use type inference algorithms and have a background in the world of programming languages (cf. [12]).

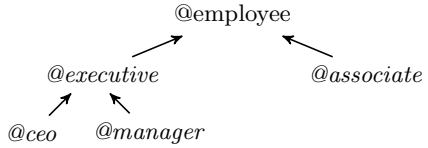
From a logical point of view, types restrict the state space of the database and therefore can be understood as constraints that each valid database instance must satisfy. In this regard, type-based query optimization is quite similar to semantic query optimization (SQO), where general constraints that are known to hold on the database instance (also called data dependencies) are used to find equivalent query rewritings, with the goal to obtain more efficient evaluation plans [24, 9, 37, 1, 39]. Beyond query optimization, constraint-based rewriting has been successfully applied in many other database areas, such as query rewriting using views [22], data exchange [17], peer data exchange [20], data integration [27], and probabilistic databases [34].

Due to their close connection, it is natural to integrate both typing knowledge and integrity constraints, such as tuple generating dependencies (TGDs) and equality generating dependencies (EGDs) [16, 3], into a single logical framework [5, 18]. In such a framework, one can fall back on established techniques, such as the classical chase algorithm [11, 24, 3], to deploy a uniform optimization process. To this aim, several authors have studied encoding the typing knowledge, mainly with the following two formalisms: Datalog (and extensions) [10, 15] and Description Logics (DL) [7, 8]. However, the lack of value creation (which can be captured by TGDs) in Datalog prevents it from being a suitable candidate for modeling integrity constraints. Although some DLs can capture TGDs, they are unable to express EGDs. There have also been attempts to integrate EGDs in DLs [41] but very strong syntactic restrictions are required to obtain decidability of reasoning. Recently, Calì et al. [6] have proposed $Datalog^{\pm}$, to extend plain Datalog with guarded TGDs and stratified negation. Yet, $Datalog^{\pm}$ can not express disjunctive rules. In order to express typing knowledge, integrity constraints containing negation and disjunction are required. To the best of our knowledge, a framework targeted at query optimization has not been investigated before.

As a motivating example, let us consider query optimization in the context of an employee database, where the following typed relations are given (types are prefixed with @):

$Person(\text{id}: @employee, \text{gender}: @gender)$
$UpperFloor(\text{id}: @ceo, \text{room}: @int)$
$MiddleFloor(\text{id}: @manager, \text{room}: @int)$
$LowerFloor(\text{id}: @associate, \text{room}: @int)$

Further assume that, in the style of an object-oriented database, the following type hierarchy is given.



*This author was funded by DFG grant LA 598/7-1.

PODS'10, June 6–11, 2010, Indianapolis, Indiana, USA.

For short, *@executive* and *@associate* are subtypes of *@employee*, and *@executive* splits up into *@ceo* and *@manager*. Further assume that (i) *@associate*, *@executive* partition @employee, and (ii) *@ceo*, *@manager* partition *@executive*.

We represent types as unary relations over the domain of the database, identified by a leading "@" symbol, e.g. write *@employee(x)* to denote that x is of type *@employee*. Suppose that, in coexistence with the restrictions imposed by the type system itself, the two constraints

$\alpha_1 := \mathit{Person}(x, y), \mathit{@ceo}(x) \rightarrow \exists z\ \mathit{UpperFloor}(x, z)$,
$\alpha_2 := \mathit{Person}(x, y), \mathit{@manager}(x) \rightarrow \exists z\ \mathit{MiddleFloor}(x, z)$

are given, enforcing that all CEOs are sitting in the upper floor and all managers are sitting in the middle floor. Having described the setting, we now turn towards query optimization. Consider the following two conjunctive queries

$q_1 : \mathit{ans}(x) \leftarrow \mathit{Person}(x, y), \mathit{UpperFloor}(x, z)$,
$q_2 : \mathit{ans}(x) \leftarrow \mathit{Person}(x, y), \mathit{MiddleFloor}(x, z)$

and assume that we are interested in computing the union $Q_{1\vee 2} := q_1 \vee q_2$, i.e. all persons sitting in the middle and upper floor. It is easy to see that, when given *only* the data dependencies α_1, α_2 or *only* the type information as input, $Q_{1\vee 2}$ is minimal w.r.t. to the number of atoms and unions in the query. Yet, when combining both typing and constraint knowledge, we can derive that $Q_{1\vee 2}$ is equivalent (on each database instance satisfying the constraints and type restrictions) to the simpler query

$q'_{1\vee 2} : \mathit{ans}(x) \leftarrow \mathit{Person}(x, y), \mathit{@executive}(x)$.

To see why, consider q_1 and first observe that $\mathit{@ceo}(x)$ must hold, due to the type restriction in the first position of relation *UpperFloor*; hence, we can add the literal $\mathit{@ceo}(x)$ to the body of q_1. But then α_1 implies that there is an entry in relation *UpperFloor* that contains x in its first position, so q_1 is equivalent to $q'_1 : \mathit{ans}(x) \leftarrow \mathit{Person}(x, y), \mathit{@ceo}(x)$. With similar argumentation, we obtain that q_2 is equivalent to $q'_2 : \mathit{ans}(x) \leftarrow \mathit{Person}(x, y), \mathit{@manager}(x)$. Given these two rewritings and the type constraint that *@executive* is exactly the union of *@ceo* and *@manager*, we conclude that $Q_{1\vee 2}$ is equivalent to $q'_{1\vee 2}$ above. Another possible rewriting for $Q_{1\vee 2}$ is the conjunctive query with (safe) negation

$q''_{1\vee 2} : \mathit{ans}(x) \leftarrow \mathit{Person}(x, y), \neg\mathit{@associate}(x)$,

because *@executive* and *@associate* partition *@employee*.

The previous example does not only show that queries may exhibit non-trivial rewritings in the presence of constraints and types, but also demonstrates that a framework that exploits data dependencies and type information at the same time may give us better optimization results than a sequential, isolated application of these information.

We implement our combined optimization approach in a logic-based framework, where we encode both the data dependencies and the type restrictions in first-order logic. To give an example, for our type hierarchy from before we use constraints like $\beta_1 := \mathit{@executive}(x) \rightarrow \mathit{@employee}(x)$ to fix the subtype relations, and may use the constraint $\beta_2 := \mathit{@executive}(x) \wedge \mathit{@associate}(x) \rightarrow \neg\mathit{@associate}(x)$ to enforce that *@associate* and *@executive* are disjoint. Following the de-facto standard approach, we then use the classical chase algorithm for the optimization process. In particular, we use a variation of the Chase & Backchase algorithm (C&B) [1], an extension of the chase developed to enumerate minimal queries in the presence of constraints.

While straightforward by idea, our approach brings along many new technical challenges, mainly due to the fact that the encoding of non-trivial type systems involves constraints containing disjunction and negation (cf. constraint β_2 above). Previous work on semantic query optimization, though, has mainly focused on tuple-generating and equality-generating dependencies, which contain neither negation nor disjunction. Here, we consider TGDs with disjunction and negation (denoted as $\text{TGD}^{\vee,\neg}$) and EGDs containing disjunction ($\text{EGD}^{\vee}$). While the chase algorithm can easily be extended to these constraint classes (cf. [13]), to date only few is known about its properties in that setting. Such properties are the central topic in this paper. In the following, we sketch the major contributions of our work in more detail.

Contributions. (1) We develop a framework that integrates type-based optimization into the semantic optimization process. In this framework, data dependencies (modeled as first-order sentences) coexist with a so-called type system, which is represented as (i) a set $\mathcal{T}$ of unary predicate symbols, one for each type, (ii) a type interpretation for constants, and (iii) a set of full constraints (i.e., constraints without existential quantification) modeling interrelations between types, such as inclusion or disjointness constraints. We are not aware of (database-related) type interrelations that cannot be encoded using this framework.

(2) On top of our framework, we present an algorithm that, in the style of the C&B algorithm from [1], can be used to optimize and minimize unions of conjunctive queries with negation w.r.t. a constraint base, a type system, and a generic cost function. Whether this framework gives us the power to compute minimal rewritings of an input query lastly depends on the termination of the underlying chase, just like in the context of standard TGDs and EGDs. As two central results, we show that, given a conjunctive query Q with union and safe negation, a constraint base Σ, a type system $\mathcal{S}$, and a generic cost function c as input, query minimization under Σ and $\mathcal{S}$ is possible whenever (i) Σ contains *full* $\text{TGD}^{\vee,\neg}$ and $\text{EGD}^{\vee}$ constraints only or (ii) Σ contains $\text{TGD}^{\vee,\neg}$ and $\text{EGD}^{\vee}$ constraints, negation in the query and constraints is restricted, and we can guarantee termination of the chase of Q with Σ. Although stated for our specific optimization framework, these results can be understood as consequent enhancements of previous results on containment testing and minimization under dependencies. The second result also improves previous results on containment testing in the context of negation from [13], which – as we will show in Section 4.1 – is essentially restricted to full TGDs.

(3) In response to the central role of chase termination in our setting, we develop novel chase termination conditions for constraint sets involving disjunction and negation. Rather than developing these termination conditions from scratch, our approach is to carry over existing sufficient termination conditions for standard TGDs and EGDs, i.e. we show how to make existing conditions applicable in the context of constraint sets involving disjunction and negation.

(4) Previous work on chase termination has focused on asserting termination for all chase sequences. We relax this restriction and identify situations where we can guarantee the existence of at least one terminating chase sequence. As we can statically derive this terminating sequence, these termination guarantees allow us to compute the chase result and therefore are of immediate practical relevance.

(5) We study the complexity of decision problems related to our type-based semantic optimization scheme. Our results confirm that – whenever we can guarantee chase ter-

mination – important problems like query equivalence or minimality testing under constraints and types w.r.t. a cost function fall into low levels of the polynomial hierarchy [40]. Referring to the experimental evaluation of the related C&B algorithm [36], we shall expect that our techniques are feasible in practice, in particular if the query is small and the number of constraints and type information is limited.

Structure. The remainder of the paper is structured as follows. We start with the preliminaries in the following subsection. In Section 3 we introduce our first-order logic based framework to semantic query optimization in the presence of types, before presenting central results in Section 4. Next, we investigate the complexity of related decision problems in Section 5. We then turn towards an investigation of chase termination in the presence of disjunction and negation in Section 6 and conclude with final remarks in Section 7.

2. PRELIMINARIES

General mathematical notation. The natural numbers $\mathbb{N}$ do not include 0. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, ..., n\}$. For a set M, we denote by $|M|$ its cardinality.

Databases. We fix three pairwise disjoint, infinite sets: the set of *constants* Δ, the set of *labeled nulls* Δ_{null}, and the set of *variables* V. Often we will denote a sequence of variables, constants or labeled nulls by $\overline{a}$ if the length of this sequence is understood from the context. A *database schema* $\mathcal{R}$ is a finite set of relational symbols. To every relational symbol $R \in \mathcal{R}$ we assign a natural number $ar(R) \geq 1$ called its arity. A database position is a pair (R, i) where $R \in \mathcal{R}$ and $i \in [ar(R)]$, for short we write R^i. Additionally, we have a set $\mathcal{T}$ of unary relational symbols that represent types for our schema. In the rest of the paper, we assume the database schema, the type symbols and the set of constants and labeled nulls to be fixed. A *database instance* I is a set of $\mathcal{R}$-atoms that contains only elements from $\Delta \cup \Delta_{null}$ in its positions. The domain of I, $dom(I)$, is the set of elements from $\Delta \cup \Delta_{null}$ appearing in I.

Homomorphisms. As usual, a homomorphism from a set of literals A_1 to a set of literals A_2 is a mapping $\mu : \Delta \cup V \to \Delta \cup \Delta_{null}$ such that the following conditions hold: (i) if $c \in \Delta$, then $\mu(c) = c$, (ii) if $R(c_1, ..., c_n) \in A_1$, then $R(\mu(c_1), ..., \mu(c_n)) \in A_2$, and (iii) if $\neg R(c_1, ..., c_n) \in A_1$, then $\neg R(\mu(c_1), ..., \mu(c_n)) \in A_2$. We write $A_1 \to A_2$ to express that there is a homomorphism from A_1 to A_2.

Conjunctive queries (with union and negation). A conjunctive query with negation ($\text{CQ}^\neg$) is an expression of the form $ans(\overline{x}) \leftarrow \varphi(\overline{x}, \overline{z})$, where φ is a conjunction of relational $\mathcal{R}$-literals, $\overline{x}$, $\overline{z}$ are sequences of variables and constants, and it holds that every variable in $\overline{x}$ also occurs in φ. We restrict our discussion to safe queries, i.e. every $\text{CQ}^\neg$ has the property that every variable that occurs in the query also occurs in some positive $\mathcal{R}$-atom. This is an easy syntactic restriction that ensures domain-independence. Whenever we speak of a query in this paper, we tacitly assume that it fulfills this safety condition. If a $\text{CQ}^\neg$ contains no negation we call it a conjunctive query CQ.

A union of conjunctive queries with negation ($\text{UCQ}^\neg$) is an expression of the form $\bigvee_{i \in [n]} q_i$, where all q_i are conjunctive queries with safe negation and all head predicates have the same arity. If a $\text{UCQ}^\neg$ contains no negation we call it a union of conjunctive queries UCQ. The result of evaluating $Q \in \text{UCQ}^\neg$ on a finite database instance I is defined as usual and denoted by $Q(I)$. If $Q, Q' \in \text{UCQ}^\neg$ we say that Q is contained in Q' ($Q \sqsubseteq Q'$) iff for all databases I it holds that $Q(I) \subseteq Q'(I)$. Q and Q' are equivalent, $Q \equiv Q'$, iff $Q \sqsubseteq Q'$ and $Q' \sqsubseteq Q$. Given a set of first-order sentences Σ, we say that Q is contained in Q' under Σ iff for all databases I s.t. I satisfies Σ it holds that $Q(I) \subseteq Q'(I)$. We write $Q \equiv_\Sigma Q'$ iff $Q \sqsubseteq_\Sigma Q'$ and $Q' \sqsubseteq_\Sigma Q$. By convention, we denote $\text{CQ}^\neg$ by lowercase and $\text{UCQ}^\neg$ by uppercase letters. We write $q \in Q$ iff q is a disjunct of Q. Abusing notation, we write $q_1 \to q_2$ iff there is a homomorphism from the set of atoms in q_1 (including the head atom) to the set of atoms in q_2 (also including the head atom). By $Q|_\tau$ we denote the query Q from which all non-τ-literals were dropped.

If $q \in \text{CQ}^\neg$, then $db(q)$ is the database that consists of one tuple for each positive atom in q, where each variable x has been replaced by constant c_x. We define $ans(q)$ to be the tuple in the head of q where again each variable x has been replaced by constant c_x. We say that q is satisfiable if there is a database instance I such that $q(I) \neq \emptyset$. Notice that q is satisfiable iff it contains no atom that appears positively and negatively in q. For $Q \in \text{UCQ}^\neg$, we set $db(Q) := \{db(q) \mid q \in Q\}$ and say that it is satisfiable iff there is a database I such that $Q(I) \neq \emptyset$.

Constraints. Let $\overline{x}, \overline{y}$ be sequences of variables. We consider two types of database constraints: *tuple-generating dependencies with union and negation* ($\text{TGD}^{\vee,\neg}$) and *equality generating dependencies with union* ($\text{EGD}^\vee$). A $\text{TGD}^{\vee,\neg}$ φ is a first-order sentence $\forall \overline{x}(\phi(\overline{x}) \to \bigvee_{i \in [n]} \exists \overline{y}_i \psi_i(\overline{x}, \overline{y}_i))$ such that (a) $\phi, \psi_1, ..., \psi_n$ are conjunctions of literals, possibly with constants, (b) $\psi_1, .., \psi_n$ are not empty, (c) ϕ is possibly empty, (d) $\phi, \psi_1, ..., \psi_n$ do not contain equality atoms and (e) for all $i \in [n]$ all variables from $\overline{x}$ that occur in ψ_i must also occur in ϕ. We obtain the classes $\text{TGD}^\neg$, $\text{TGD}^\vee$, and TGD by disallowing $\vee$, $\neg$, and both $\vee$ and $\neg$. By φ^i we denote the TGD $\forall \overline{x}(\phi(\overline{x}) \to \exists \overline{y}_i \psi_i(\overline{x}, \overline{y}_i))$ and set $\widehat{\varphi} := \{\varphi_1, ..., \varphi_n\}$.

An $\text{EGD}^\vee$ φ is a first-order logic sentence of the form $\forall \overline{x}(\phi(\overline{x}) \to \bigvee_{i \in [n]} x_{i,1} = x_{i,2})$, where all $x_{i,1}, x_{i,2}$ occur in ϕ and ϕ is a non-empty conjunction of equality-free $\mathcal{R}$-atoms, possibly with constants. We obtain the subclass EGD from $\text{EGD}^\vee$ by disallowing $\vee$ in the conclusion. By φ^i we denote the EGD $\forall \overline{x}(\phi(\overline{x}) \to x_{i,1} = x_{i,2})$ and set $\widehat{\varphi} := \{\varphi_1, ..., \varphi_n\}$.

When using the word "constraints" in the following, we always mean the union of the classes $\text{TGD}^{\vee,\neg}$ and $\text{EGD}^\vee$. Satisfaction of constraints by databases is defined in the standard first-order manner. We write $I \models \alpha$ if a constraint α is satisfied by I and $I \not\models \alpha$ otherwise. As a notational convenience, we will often omit the $\forall$-quantifier and the respective list of universally quantified variables. A set of constraints is full if it contains no existentially quantified variables.

We use the term $body(\alpha)$ for a constraint α as the set of atoms in its premise; analogously $head(\alpha)$ is the set of sets of all atoms in some disjunct of the constraint's conclusion. By $\Sigma|_\tau$ we denote the set Σ from which all non-τ-literals in the constraint bodies were dropped. If α is a constraint and $\overline{a}$ is a sequence of labeled nulls and constants, then $\alpha(\overline{a})$ is the constraint α without universal quantifiers but with parameters $\overline{a}$. We shall abuse this notation and say that a labeled null occurs in $\alpha(\overline{a})$, meaning that a labeled null is the parameter for some universally quantified variable in α.

Chase steps. Let $q \in \text{CQ}^\neg$and $\alpha \in \text{TGD}^{\vee,\neg}$ of the form $\phi_1(\overline{x}) \to \bigvee_{i \in [n]} \exists \overline{y}_i \psi_i(\overline{x}, \overline{y}_i)$. We say that α is applicable to q if there is a homomorphism μ from $body(\alpha)$ to q and for every $A \in head(\alpha)$ it holds that μ cannot be extended to a homomorphism $\mu' \supseteq \mu$ from A to q. In such a case the chase step

$q \xrightarrow{\alpha,\mu(\overline{x})} q_u$ is defined as follows. For every $i \in [n]$ we define a homomorphism ν_i as follows: (i) ν_i agrees with μ on all universally quantified variables in α, (ii) for every existentially quantified variable y in ψ_i we choose a "fresh" labeled null $n_{y,i} \in \Delta_{null} \backslash dom(db(q))$ and define $\nu(y) := n_{y,i}$. We set q_u to be the union of safe conjunctive queries with negation $\bigvee_{i\in[n]} db(q) \wedge \nu_i(\psi_i)$ from which all unsatisfiable disjuncts are removed. In case that this results in the empty query the result of the chase step is FALSE.

A chase step with an EGD is just a standard chase step in case the query is satisfiable, otherwise the result of the chase step is the empty query FALSE.

Finally, we lift chase steps to apply to unions of conjunctive queries. For a union of safe conjunctive queries with negation $q := \bigvee_{i\in[l]} q_i$ we write $q \xrightarrow{\alpha,\mu(\overline{x})} q'$ iff there is $i \in [l]$ such that $q_i \xrightarrow{\alpha,\mu(\overline{x})} q_u$ is defined and $q' := q_u \vee \bigvee_{j\in[l]\backslash\{i\}} q_j$.

Chase sequences. A chase sequence is defined analogously to standard chase sequences using the chase steps defined before, i.e. it is an exhaustive application of chase steps until no more chase step is applicable. Note that different orders of constraint application may lead to a different chase result. However, as proved in [13], two different terminating chase orders that do not fail lead to homomorphically equivalent results. Therefore, we write Q^{Σ} for the result of the chase on $Q \in \text{UCQ}^{\neg}$ under constraints Σ. In case that the constraint set involves disjunction, a chase sequence can be represented as a chase tree, as in [17]. The disjunction of the tree's leaves represents the resulting query.

Chase termination. In the course of this paper, we investigate two flavors of chase termination, defined as follows.

- $\text{CT}_{\forall\forall} := \{\Sigma \mid$ for every finite database instance I and for every chase sequence the chase with Σ and I terminates$\}$
- $\text{CT}_{\forall\exists} := \{\Sigma \mid$ for every finite database instance I there exists a terminating chase sequence with $\Sigma\}$

If $\Sigma \in \text{CT}_{\forall\exists}$ we can compute a terminating chase sequence by applying the chase in a breadth-first manner, i.e. generating a tree whose root is the start instance, its children are obtained by applying one chase step on the start instance and the tree is expanded in breadth-first manner.

Canonical databases and completeness. We say that a query $q \in \text{CQ}^{\neg}$ is complete [13] iff it is satisfiable and for all $q' \in \text{CQ}^{\neg}$ it holds that $ans(q) \in q'(db(q))$ implies $q' \to q$. $Q \in \text{UCQ}^{\neg}$ is complete iff all disjuncts are complete. It was shown in [13] that for every $Q \in \text{UCQ}^{\neg}$ we can compute a unique $Q' \in \text{UCQ}^{\neg}$ that is complete and $Q \equiv Q'$. Therefore, we denote Q' by $comp(Q)$. Let $ADom \notin \mathcal{R} \cup \mathcal{T}$. It was shown in [13] that $comp(Q) = Q^{\Sigma_{\neg}}|_{\mathcal{R}\cup\mathcal{T}}$, where we define $\Sigma_{\neg}$ as the set which contains for every $R \in \mathcal{R} \cup \mathcal{T}$ with $ar(R) = k$ the constraints

$ADom(x_1), ..., ADom(x_k) \to R(x_1, ..., x_k) \vee \neg R(x_1, ..., x_k)$
$R(x_1, ..., x_k) \to ADom(x_1), \ldots, ADom(x_k)$.

For $q \in \text{CQ}^{\neg}$, we use $size(q)$ as an abbreviation for the number of literals in q. We extend $size(Q)$ to $Q \in \text{UCQ}^{\neg}$ by $size(Q) := \Sigma_{q\in Q} size(q)$.

3. CONSTRAINTS AND TYPES

Our approach to combined semantic and type-based optimization relies on a rigorous first-order logic formalization. Given our relational schema $\mathcal{R}$, we assume a special vocabulary $\mathcal{T}$ consisting of unary relation symbols. For $a \in \Delta$ we define its associated type interpretation $\text{TYPE}(a)$ which is a set of literals that exactly contains for every $T \in \mathcal{T}$ either $T(a)$ or $\neg T(a)$. Abusing notation we identify T with the set $\{a \in \Delta \mid T(a) \in \text{TYPE}(a)\}$ and analogously write $\neg T$ instead of $\{a \in \Delta \mid \neg T(a) \in \text{TYPE}(a)\}$. A type hierarchy H is a set of full constraints over the schema $\mathcal{T}$. A type system over $\mathcal{T}$ is a tuple (H, TYPE). We are interested in type systems in which the type hierarchy adheres to the type interpretation according to the following definition[1].

DEFINITION 1. *We say that a type hierarchy H reflects* TYPE *if H is logically equivalent to the set of constraints obtained as follows: for all $1 \le k, l \le |\mathcal{T}|$ and for all $A_1, ..., A_k$, $B_1, ..., B_l \in \{T, \neg T \mid T \in \mathcal{T}\}$,*

- *if $\emptyset \neq A_1 \cap ... \cap A_k \subseteq B_1 \cup ... \cup B_l$, we have a full constraint of the form $A_1(x), ..., A_k(x) \to B_1(x) \vee ... \vee B_l(x)$,*
- *if $\emptyset \neq A_1 \cap ... \cap A_k$ is finite, we have an $\text{EGD}^{\vee}$ of the form $A_1(x), ..., A_k(x) \to \bigvee_{a\in A_1\cap...\cap A_k} x = a$, and*
- *if $\emptyset = A_1 \cap ... \cap A_k$, a TGD $A_1(x), ..., A_k(x) \to \neg A_1(x)$.* □

Informally speaking, a type system reflects a type interpretation if (i) all subsumption relationships between types can be derived from the type hierarchy, (ii) whenever a type or the intersection of several types is finite but non-empty, there is a constraint that fixes the domain of the type, and (iii) whenever two types are disjoint, we can derive this information using the constraints in the type hierarchy. The following example illustrates the previous definition.

EXAMPLE 1. *We formalize the type system of our motivating example from the Introduction. First, we define the vocabulary $\mathcal{T}_1 := \{$@employee, @manager, @ceo, ...$\}$, where we interpret the elements of $\mathcal{T}_1$ as unary relation symbols. We then fix a type interpretation TYPE_1 that reflects the type relationships that were informally discussed in the Introduction. Consider for instance the constant a_1 := 'CEO1' standing for a CEO. Its interpretation is defined as*

$$\begin{aligned}\text{TYPE}_1(a_1) := \{ &@ceo(a_1), @executive(a_1), @employee(a_1),\\ &\neg @manager(a_1), \neg @associate(a_1),\\ &\neg @gender(a_1), \neg @int(a_1)\},\end{aligned}$$

stating that a_1 is of type @ceo, @executive, and @employee, but not of type @manager, @associate, and so on. The type hierarchy H_1, which we define as

$$\begin{aligned}H_1 := \{ &\beta_1 := @ceo(x) \to @executive(x),\\ &\beta_2 := @manager(x) \to @executive(x),\\ &\beta_3 := @executive(x) \to @employee(x),\\ &\beta_4 := @associate(x) \to @employee(x),\\ &\beta_5 := @gender(x) \to x = \text{'male'} \vee x = \text{'female'},\\ &\beta_6 := @manager(x), @ceo(x) \to \neg @manager(x),\\ &\beta_7 := @executive(x), @associate(x) \to \neg @executive(x),\\ &\beta_8 := @employee(x), @gender(x) \to \neg @employee(x),\\ &\beta_0 := @employee(x), @int(x) \to \neg @employee(x),\\ &\beta_{10} := @gender(x), @int(x) \to \neg @gender(x)\},\end{aligned}$$

satisfies Definition 1: β_1 – β_4 model all type subsumption relationships (cf. bullet one of the definition). Next, β_5 fixes the domain of type @gender (cf. bullet two), which we assume to be the only finite type in our scenario. Finally, constraints β_6 – β_{10} express disjointness between incompatible types (cf. bullet three). Observe that H_1 implicitly contains other relationships, for instance the constraint @ceo(x) → @employee(x) can be derived from β_1 and β_3; this is in line with Definition 1, which enforces only logical equivalence to a "complete" list of constraints. □

[1]Definition 1 must not be understood as an algorithm. It provides means for a database schema designer to represent a type hierarchy as a set of constraints.

Our framework is able to model object-oriented features. Object-oriented Datalog can be reduced to normal Datalog with negation with a compilation scheme as in [2].

Having established a framework to express type hierarchies in first-order logic, we now extend it to general data dependencies used in semantic query optimization.

DEFINITION 2. *We call a tuple* (Σ, H, TYPE) *a typed relational schema iff (i)* Σ *is a set of integrity constraints over* $\mathcal{R} \cup \mathcal{T}$*, (ii)* Σ *contains no negative* $\mathcal{R}$*-literals, (iii) for all* $\alpha \in \Sigma$ *it holds that every variable in* α *appears in some* $\mathcal{R}$*-atom, and (iv)* (H, TYPE) *is a type system over* $\mathcal{T}$ *such that* H *reflects* TYPE. □

Note that Σ is defined over $\mathcal{R} \cup \mathcal{T}$, so it may contain both data dependencies in the common sense and dependencies involving types. In particular, we can use Σ to encode type information implicitly given by the schema:

EXAMPLE 2. *To capture our example scenario from the Introduction, we define* $\Sigma_1 := \{\alpha_1, \alpha_2, \gamma_1, \gamma_2, \gamma_3, \gamma_3\}$*, where* α_1 *and* α_2 *are the constraints from the Introduction asserting that CEOs are sitting in the upper floor and managers are sitting in the middle floor and* γ_1 *–* γ_4 *are defined as*

$\gamma_1 := \mathit{Person}(x, y) \rightarrow @\mathit{employee}(x), @\mathit{gender}(y),$
$\gamma_2 := \mathit{UpperFloor}(x, y) \rightarrow @\mathit{ceo}(x), @\mathit{int}(y),$
$\gamma_3 := \mathit{MiddleFloor}(x, y) \rightarrow @\mathit{manager}(x), @\mathit{int}(y),$
$\gamma_4 := \mathit{LowerFloor}(x, y) \rightarrow @\mathit{associate}(x), @\mathit{int}(y).$

Using TYPE_1 *and* H_1 *introduced in Example 1, the tuple* $\mathcal{S}_1 := (\Sigma_1, H_1, \text{TYPE}_1)$ *is a typed relational schema.* □

Given a typed relational schema $\mathcal{S} := (C_1, C_2, C_3)$, we will use the conventions that $\Sigma(\mathcal{S}) := C_1$, $H(\mathcal{S}) := C_2$, and $\text{TYPE}(\mathcal{S}) := C_3$. The following definition of satisfying database instances is straightforward.

DEFINITION 3. *An* $\mathcal{R} \cup \mathcal{T}$*-database instance* I *satisfies a typed relational schema* $\mathcal{S}$*,* $I \models \mathcal{S}$*, iff* $I|_{\mathcal{R}} \neq \emptyset$*, for every constant* $a \in \mathit{dom}(I) \cap \Delta$ *we have that* $I \models \text{TYPE}(\mathcal{S})(a)$*, and it holds that* $I \models \Sigma(\mathcal{S}) \cup H(\mathcal{S})$. □

We are now in the position to define the notions of query containment, equivalence, and minimality in typed schemas.

DEFINITION 4. *Let* $\mathcal{S}$ *be a typed relational schema. For* $Q, Q' \in \text{UCQ}^{\neg}$*, we write* $Q \sqsubseteq_{\mathcal{S}} Q'$ *iff for all* $I \models \mathcal{S}$*, we have that* $Q(I) \subseteq Q'(I)$*.* Q, Q' *are equivalent under* $\mathcal{S}$*,* $Q \equiv_{\mathcal{S}} Q'$*, iff* $Q \sqsubseteq_{\mathcal{S}} Q'$ *and* $Q' \sqsubseteq_{\mathcal{S}} Q$. □

As we are not aware of a generally accepted notion of minimality for the class $\text{UCQ}^{\neg}$, we abstract from a concrete cost measure and use a generic cost function instead.

DEFINITION 5. *Let* $\mathcal{L} \in \{\text{UCQ}^{\neg}, \text{UCQ}, \text{CQ}^{\neg}, \text{CQ}\}$ *be a query language. A cost function for* $\mathcal{L}$ *is a polynomial-time computable* $c : \mathcal{L} \rightarrow \mathbb{N}$ *such that* $\mathit{size}(Q) \leq c(Q)$ *and for every subquery* $\mathit{sub} \subseteq Q$ *we have that* $c(\mathit{sub}) \leq c(Q)$*. Given a typed relational schema* $\mathcal{S}$ *and a query* $Q \in \mathcal{L}$*, we say that* Q *is* $(\mathcal{L}, c, \mathcal{S})$*-minimal iff there is no* $Q' \in \mathcal{L}$ *such that* $Q \equiv_{\mathcal{S}} Q'$ *and* $c(Q') \leq c(Q)$*. We say that* Q' *is an* $(\mathcal{L}, c, \mathcal{S})$*-rewriting of* Q *iff* $Q' \equiv_{\mathcal{S}} Q$ *and* $c(Q') < c(Q)$*. An* $(\mathcal{L}', c, \mathcal{S})$*-minimal rewriting of* Q *is a query* $Q' \in \mathcal{L}'$ *such that* Q' *is* $(\mathcal{L}', c, \mathcal{S})$*-minimal and* $Q \equiv_{\mathcal{S}} Q'$. □

It is easily shown that the generic cost function imposes an upper bound on the size of minimal rewritings:

PROPOSITION 1. *Let* $Q \in \text{UCQ}$*. The set of* $Q' \in \text{UCQ}^{\neg}$ *with* $c(Q') \leq c(Q)$ *is finite and its size can be bounded by* $ar(\mathcal{R} \cup \mathcal{T}) \cdot c(Q)^3 \cdot (3 \cdot |\mathcal{R} \cup \mathcal{T}|)^{c(Q)} \cdot ar(\mathcal{R} \cup \mathcal{T}) \cdot (c(Q) \cdot ar(\mathcal{R} \cup \mathcal{T}) + |\mathit{dom}(Q)| + |\mathit{dom}(\Sigma(\mathcal{S}) \cup H(\mathcal{S}))|)$*. Thus, the set of* $(\text{UCQ}^{\neg}, c, \mathcal{S})$*-minimal rewritings of* Q *is also bounded by this number.* □

We conclude this section with an example that illustrates the cost function, rewritings, and minimality.

EXAMPLE 3. *Let* $Q := q_1 \vee \cdots \vee q_n$ *be a* $\text{UCQ}^{\neg}$*. Let* $\mathit{pos}(q_i)$ *denote the number of positive literals and* $\mathit{neg}(q_i)$ *the number of negative literals in the body of* q_i*. We exemplarily consider the cost function* $c_1(Q) := \sum_{1 \leq i \leq n} (\mathit{pos}(q_i) + 2 * \mathit{neg}(q_i))$*.*

Given query $Q_{1 \vee 2}$ *from the Introduction and the typed relational schema* $\mathcal{S}_1$ *from Example 2, we have that both* $q'_{1 \vee 2}$ *and* $q''_{1 \vee 2}$ *from the Introduction are* $(\text{UCQ}^{\neg}, c_1, \mathcal{S}_1)$*-rewritings of* $Q_{1 \vee 2}$*: it is easily verified that* $Q_{1 \vee 2} \equiv_{\mathcal{S}_1} q'_{1 \vee 2} \equiv_{\mathcal{S}_1} q''_{1 \vee 2}$ *and we have* $c_1(Q_{1 \vee 2}) = 4 > c_1(q''_{1 \vee 2}) = 3 > c_1(q'_{1 \vee 2}) = 2$*.*

By enumerating all candidate $(\text{UCQ}^{\neg}, c_1, \mathcal{S}_1)$*-rewritings of* $Q_{1 \vee 2}$ *it can be shown that* $q'_{1 \vee 2}$ *is* $(\text{UCQ}^{\neg}, c_1, \mathcal{S}_1)$*-minimal.* □

4. SEMANTIC QUERY OPTIMIZATION IN TYPED RELATIONAL SCHEMAS

Having presented our framework, the goal of this section is to develop rewriting techniques and to identify fragments for which rewritings (and hence, by Proposition 1, also minimal rewritings) can be computed. Note that, in the general case, query containment for conjunctive queries under TGDs and EGDs is already undecidable,so it follows immediately that query containment in our fragment (where we consider extended classes of TGDs and EGDs, as well as CQs with union and negation) is generally undecidable. Therefore, the study of decidable fragments is of high practical interest.

In our effort to provide a mechanism for containment testing, we start with a slight variant of the standard chase algorithm which, after each chase step, adds type information for constants that were introduced during the chase.

DEFINITION 6. *Let* $Q \in \text{UCQ}^{\neg}$*. The sequence* $(Q_i)_{i \in \mathbb{N}}$ *is inductively defined as follows.* $Q_1 := Q$*. For* $i \geq 2$ *we set* $Q_i := \bigvee_{q \in Q_{i-1}^{\Sigma(\mathcal{S}) \cup H(\mathcal{S})}} q'$*, where* q' *is defined as* q *to which* $\text{TYPE}(a)$ *has been added to the body of the query for all* $a \in \Delta \cap \mathit{dom}(db(q))$*. If there is some* $i \in \mathbb{N}$ *such that* $Q_i = Q_{i+1}$*, then* $Q^{\mathcal{S}} := Q_i$. □

EXAMPLE 4. *Consider* $\mathcal{S}_1 := (\Sigma_1, H_1, \text{TYPE}_1)$ *from Example 2, query* $q : \mathit{ans}() \leftarrow \mathit{Person}(\text{'CEO1'}, y)$*, and assume that* $@\mathit{ceo}(\text{'CEO1'}) \in \text{TYPE}_1(\text{'CEO1'})$*. When chasing* q *according to Definition 6, we obtain the query* $q^{\mathcal{S}}$ *defined as*

$$\begin{aligned} \mathit{ans}() \leftarrow\ & \mathit{Person}(\text{'CEO1'}, y), \mathit{UpperFloor}(\text{'CEO1'}, z), \\ & @\mathit{ceo}(\text{'CEO1'}), \neg @\mathit{ceo}(y), \neg @\mathit{ceo}(z), \\ & @\mathit{manager}(\text{'CEO1'}), \neg @\mathit{manager}(y), \neg @\mathit{manager}(z), \\ & \ldots \end{aligned}$$

where the rest of the query contains some more type information. The interesting thing here is that, by adding the type restriction $@\mathit{ceo}(\text{'CEO1'})$ *according to our modified version of the chase, we obtain precise type information for the constant 'CEO1' and therefore in subsequent chase steps are able to derive the literal* $\mathit{UpperFloor}(\text{'CEO1'}, z)$ *(which is implied by the constraint* α_1 *from the Introduction).* □

It can be shown that our modified chase terminates whenever the standard chase algorithm terminates:

PROPOSITION 2. *Let $\mathcal{S}$ be fixed. If the mapping $Q \mapsto Q^{\Sigma(\mathcal{S})\cup H(\mathcal{S})}$ can be computed in polynomial time, then $Q^{\mathcal{S}}$ is defined and the mapping $Q \mapsto Q^{\mathcal{S}}$ can be computed in polynomial time.* □

In the following, we show that our modified chase with $\Sigma(\mathcal{S})$ and $H(\mathcal{S})$ indeed gives universal models and therefore always queries that are equivalent under $\mathcal{S}$.

DEFINITION 7. *We call a finite set of database instances $\{I_1, ..., I_n\}$ universal for a typed relational schema $\mathcal{S}$ and a set of database instance $\mathcal{J}$ iff (i) for all $i \in [n]$: $I_i \models \mathcal{S}$, and (ii) for every database instance I it holds that if $I \models \mathcal{S}$ and there is some $J \in \mathcal{J}$ such that $J \to I$, then there is $i \in [n]$ such that $I_i \to I$.* □

LEMMA 1. *Let $\mathcal{S}$ be a typed relational schema, $Q \in \text{UCQ}^\neg$. If $Q^{\mathcal{S}}$ exists, then $db(Q^{\mathcal{S}})$ is universal for $(\mathcal{S}, db(Q))$.* □

LEMMA 2. *Let $Q \in \text{UCQ}^\neg$. It holds that $Q \equiv_{\mathcal{S}} Q^{\mathcal{S}}$.* □

The central idea of our approach now is to use the modified chase from Definition 6 to find minimal rewritings. More precisely, given a query Q, typed relational schema $\mathcal{S}$, and cost function c, we first compute $Q^{\mathcal{S}}$, enumerate all candidate candidate rewritings Q_i (whose number is bounded by Proposition 1), and finally check if $Q \equiv_{\mathcal{S}} Q_i$ holds. Note that, compared to the C&B algorithm from [1], we lose the property that every minimal rewriting Q' of Q is a subquery of $Q^{\mathcal{S}}$, as witnessed by the following example.

EXAMPLE 5. *Consider the union of the two conjunctive queries $p_1 : ans(x) \leftarrow Person(x, \text{'male'}), militaryService(x)$ and $p_2 : ans(x) \leftarrow Person(x, \text{'female'})$, extracting all men who have completed their compulsory military service and all women. Let $\delta := Person(x, \text{'male'}) \to militaryService(x)$ and consider $\mathcal{S}_1$, c_1 from Example 3. Given that position $Person^2$ is typed with @gender and $H(\mathcal{S}_1)$ contains the constraint $\beta_5 := @gender(x) \to x = \text{'male'} \vee x = \text{'female'}$, it follows that $ans(x) \leftarrow Person(x, y)$ is a minimal rewriting of $p_1 \vee p_2$ w.r.t. $(\text{UCQ}^\neg, c_1, (\{\delta\}, H(\mathcal{S}_1), \text{TYPE}(\mathcal{S}_1))$.* □

The only reason that may prevent us from computing (minimal) rewritings in typed schemas is that the chase (for the original query or the rewriting candidate queries) does not necessarily terminate. In the remainder of this section, we therefore investigate decidable fragments.

4.1 Fragment I

We first carry over the results on containment testing for conjunctive queries in the presence of negation from [13] into the context of typed relational schemas. As a side contribution, we show that this approach works for full TGDs only.

Let $\mathcal{S} \cup \Sigma_\neg$ denote the typed relational schema in which $\Sigma_\neg$ has been added to $\Sigma(\mathcal{S})$. The next lemma transfers Theorem 9 from [13] into the context of typed relational schema:

LEMMA 3. *Let $Q, Q' \in \text{UCQ}^\neg$. If $Q^{\mathcal{S}\cup\Sigma_\neg}$ exists, then it holds that $Q \sqsubseteq_{\mathcal{S}} Q'$ iff for every $P \in Q^{\mathcal{S}\cup\Sigma_\neg}|_{\mathcal{R}\cup\mathcal{T}}$ there is $P' \in Q'$ such that $P' \to P$.* □

From this lemma we obtain the following theorem, which – in combination with the chase – gives us a query minimization algorithm whenever the chase with $\mathcal{S} \cup \Sigma_\neg$ terminates:

THEOREM 1. *There is an algorithm that, given $Q \in \text{UCQ}^\neg$ such that $Q \mapsto Q^{\mathcal{S}\cup\Sigma_\neg}$ is computable, enumerates exactly all $(\text{UCQ}^\neg, c, \mathcal{S})$-minimal rewritings of Q up to isomorphism.* □

The problem with this result, though, is that the extension of the constraint set by $\Sigma_\neg$ often leads to non-terminating chase sequences, even if the chase with $\mathcal{S}$ terminates.

EXAMPLE 6. *If $\Sigma(\mathcal{S}) := \{\epsilon_1 := R(x_1, x_2) \to \exists y E(x_2, y)\}$ and $q() := R(x_1, x_2)$, then $q^{\mathcal{S}\cup\Sigma_\neg}$ is not defined. To see why, observe that $\Sigma_\neg$ contains (amongst others) the constraints*

$$\epsilon_2 := E(x_1, x_2) \to ADom(x_1), ADom(x_2),$$
$$\epsilon_3 := ADom(x_1), ADom(x_2) \to R(x_1, x_2) \vee \neg R(x_1, x_2).$$

It is easily verified that there is no terminating chase sequence for the chase of q with $\{\epsilon_1, \epsilon_2, \epsilon_3\}$. The same still holds when chasing with the full set $\Sigma(\mathcal{S}) \cup \Sigma_\neg$. □

In the light of the previous example, there is only little hope that the chase algorithm terminates when Σ contains TGDs with existentially quantified variables. With this observation in mind, the next corollary identifies the only situation where Theorem 1 is of practical benefit:

COROLLARY 1. *Let $\Sigma(\mathcal{S})$ consist of full constraints only. There is an algorithm that, given $\mathcal{S}$ and $Q \in \text{UCQ}^\neg$ as input, enumerates exactly all $(\text{UCQ}^\neg, c, \mathcal{S})$-minimal rewritings of Q up to isomorphism.* □

Our finding that the above approach (and hence, also the approach proposed in [13]) is essentially restricted to full constraints motivates the search for fragments in which the minimization problem can be solved without adding $\Sigma_\neg$. We will present such a fragment in the next subsection.

4.2 Fragment II

We now define a fragment of $\text{UCQ}^\neg$. We call $Q \in \text{UCQ}^\neg$ semi-positive, denoted as $\text{UCQ}^{\neg/2}$, iff all variables that occur in a negative $\mathcal{R}$-literal occur also in the head predicate. Semi-positive queries are interesting because containment testing using the chase can be done without adding $\Sigma_\neg$:

THEOREM 2. *Let $\Sigma \subseteq TGD^{\vee,\neg} \cup EGD^\vee$, $Q \in \text{UCQ}^{\neg/2}$ without negative $\mathcal{T}$-literals and $Q' \in \text{UCQ}^\neg$. W.l.o.g. every $q \in Q'$ contains all constants from Q, Q' and Σ. If Q'^{Σ} exists, then it holds that $Q' \sqsubseteq_\Sigma Q \iff$ for all $P' \in comp(Q')^\Sigma$ there exists $P \in Q$ such that $P \to P' \iff Q'^\Sigma \sqsubseteq Q$.* □

As a corollary from the previous theorem we obtain the following result for typed relational schemas, clarifying that the constraints in $H(\mathcal{S})$ do not harm this nice property.

COROLLARY 2. *Let $\Sigma(\mathcal{S}) \subseteq \text{TGD}^{\vee,\neg} \cup \text{EGD}^\vee$, $Q \in \text{UCQ}^{\neg/2}$ without negative $\mathcal{T}$-literals and $Q' \in \text{UCQ}^\neg$. W.l.o.g. every $q \in Q'$ contains all constants from Q, Q' and $\Sigma(\mathcal{S})$. If $Q'^{\mathcal{S}}$ exists, then $Q' \sqsubseteq_{\mathcal{S}} Q \iff$ for all $P' \in comp(Q')^{\mathcal{S}}$ there exists $P \in Q$ such that $P \to P' \iff Q'^{\mathcal{S}} \sqsubseteq Q$.* □

In order to demonstrate why the condition that every $q \in Q'$ contains all constants from Q, Q' and $\Sigma(\mathcal{S})$ is not a restriction we have a look at the next example.

EXAMPLE 7. *Let $q' : ans() \leftarrow A(x, y)$ and $q : ans() \leftarrow A(x, a)$, where we want to test for $q' \sqsubseteq_{\mathcal{S}} q$. The constant a occurs in q but not in q', which contradicts our condition. We rewrite q' to $Q' : q_1 \vee q_2$, where $q_1 : ans() \leftarrow A(x, y), A(a, a)$ and $q_2 : ans() \leftarrow A(x, y), \neg A(a, a)$. Obviously, $q' \equiv Q'$. We can now proceed by testing for $Q' \sqsubseteq_{\mathcal{S}} q$.*

Clearly, the above results are applicable in practice whenever $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$ is in $\mathrm{CT}_{\forall\forall}$ or $\mathrm{CT}_{\forall\exists}$. The following lemma gives an even weaker precondition, showing cases when the constraints in $H(\mathcal{S})$ do not affect chase termination:

LEMMA 4. *For $Q \in \mathrm{UCQ}^{\neg}$: $Q \mapsto Q^{\mathcal{S}}$ is computable if $\Sigma(\mathcal{S})|_{\mathcal{R}} \in \mathrm{CT}_{\forall\forall}$, or $\Sigma(\mathcal{S}) = \Sigma(\mathcal{S})|_{\mathcal{R}} \in \mathrm{CT}_{\forall\exists}$.* □

As our central result, we obtain an algorithm for $\mathrm{UCQ}^{\neg/2}$ minimization whenever we can guarantee the termination of the underlying chase according to the previous lemma:

THEOREM 3. *There is an algorithm that, given a query $Q \in \mathrm{UCQ}^{\neg/2}$ such that $Q \mapsto Q^{\mathcal{S}}$ is computable, enumerates exactly all $(\mathrm{UCQ}^{\neg/2}, c, \mathcal{S})$-minimal rewritings of Q under Σ up to isomorphism.* □

5. COMPLEXITY

5.1 Satisfiability

As a first problem in our complexity study, we investigate the satisfiability of typed relational schemas. We define satisfiability and the associated decision problem as follows.

DEFINITION 8. *A typed relational schema $\mathcal{S}$ is satisfiable iff there is a finite $\mathcal{R} \cup \mathcal{T}$-database instance I s.t. $I \models \mathcal{S}$.* □

SATISFIABILITY:	
Input :	A typed relational schema $\mathcal{S}$.
Question:	Is there a finite instance I: $I \models \mathcal{S}$?
Answer:	Yes or no.

It turns out that, whenever there is a terminating chase sequence for the constraints induced by the data dependencies and the type hierarchy, we can use the chase algorithm to test whether a satisfying model exists or not:

THEOREM 4. *Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$. There is an algorithm that, given a typed relational schema $\mathcal{S}$ as input, decides whether $\mathcal{S}$ is satisfiable.* □

A reduction from CNF-SAT gives us an NP lower bound:

THEOREM 5. SATISFIABILITY *is* NP*-hard. This still holds if the set of integrity constraints $\Sigma(\mathcal{S})$ contains no negation and no* $\mathrm{EGD}^{\vee}$. □

The next theorem identifies a large class of typed relational schemas for which we obtain a Σ_2^P upper bound:

THEOREM 6. *Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$ ensuring polynomial depth of the chase tree. Then* SATISFIABILITY $\in \Sigma_2^P$. □

We obtain even better bounds when restricting the size of the constraint's heads:

COROLLARY 3. *Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$ ensuring polynomial depth of the chase tree and assume that the size of the head of every $\alpha \in \Sigma(\mathcal{S}) \cup H(\mathcal{S})$ is bounded by some constant k. Then* SATISFIABILITY *is in* NP. □

Observe that the previous (and some of the following) results rely on the polynomial depth of the chase tree. We will come back to this issue in Section 6, where we indicate chase termination conditions that guarantee this property.

5.2 Query Optimization

We now come to the central complexity results of this paper, namely the complexity of testing whether a query is a rewriting or a minimal rewriting of another query. We define these two decision problems as follows.

MINIMAL$(\mathcal{L}, c, \mathcal{S})$:	
Input :	$Q \in \mathcal{L}$.
Question:	Is Q $(\mathcal{L}, c, \mathcal{S})$-minimal?
Answer:	Yes or no.

REWRITE$(\mathcal{L}, c, \mathcal{S})$:	
Input :	$(Q, Q') \in \mathcal{L} \times \mathcal{L}$.
Question:	Is Q' an $(\mathcal{L}, c, \mathcal{S})$-rewriting of Q?
Answer:	Yes or no.

By definition, Q' is an $(\mathcal{L}, c, \mathcal{S})$-minimal rewriting of Q iff $Q' \in$ MINIMAL$(\mathcal{L}, c, \mathcal{S})$ and $(Q, Q') \in$ REWRITE$(\mathcal{L}, c, \mathcal{S})$. A reduction from the containment problem for conjunctive queries with inequalities gives us a lower bound for the REWRITE problem in the general case.

THEOREM 7. REWRITE$(\mathrm{CQ}, c, \mathcal{S})$ *is Π_2^P-hard.* □

Whenever the chase tree is of polynomial depth, we can also guarantee membership in Π_2^P:

THEOREM 8. *Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$ ensuring polynomial depth of the chase tree. Then,* REWRITE$(\mathrm{UCQ}^{\neg/2}, c, \mathcal{S})$ *is Π_2^P-complete.* □

Having discussed REWRITE, we conclude our complexity study with a similar results for the minimization problem:

THEOREM 9. *Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$ ensure polynomial depth of the chase tree. Then,* MINIMAL$(\mathrm{UCQ}^{\neg/2}, c, \mathcal{S}) \in \Pi_3^P$. □

To put our results into context, we point out that containment testing under negation- and disjunction-free TGDs and EGDs is already NP-hard. Nevertheless, the experiments in [36] confirm that minimization works well in practice, last but not least because the constraint sets are usually small. What we have shown here is that semantic query optimization in the presence of types, $\mathrm{TGD}^{\neg,\vee}$, and $\mathrm{EGD}^{\vee}$ falls into the lower levels of the polynomial hierarchy.

6. CHASE TERMINATION

We have seen in Section 4 that the applicability of our framework for semantic query optimization in the presence of types heavily depends on the termination of the underlying chase algorithm. In the past, there has been much research on the termination of the chase with TGDs and EGDs only, see e.g. [17, 14, 30, 31]. However, little attention has been spent on chase termination in the presence of negation and disjunction in the constraints. Notable exceptions are [17], which treats the case of TGDs and $\mathrm{EGD}^{\vee}$s, and [13], which uses an adaption of the weak acyclicity condition from [17] to allow for negation and disjunction. We present a more general approach and eliminate negation and disjunction in the constraints such that, if a termination guarantee for the rewritten constraint set can be made, then a guarantee can be also made for the original constraint set.

This allows us to reduce the problem of chase termination to a standard setting involving only TGDs and EGDs.

In addition, we study $\mathrm{CT}_{\forall\exists}$, which ensures the existence of at least one terminating chase sequence for every database instance. The literature on the chase lacks a systematic study of $\mathrm{CT}_{\forall\exists}$ and concentrates solely on $\mathrm{CT}_{\forall\forall}$. We fill this gap by identifying decidable fragments of $\mathrm{CT}_{\forall\exists}$. To the best of our knowledge, this is the first study of sufficient termination conditions for the chase which does not ensure the termination of all but at least one chase sequence. The results of our study on sequence-dependent chase termination have an important additional property. We cannot only ensure that there is a terminating chase sequence, but we can determine it statically, while checking our termination conditions. This has an important implication. We do not have to apply the chase in breadth-first, but in the usual depth-first manner, thus saving much time and space.

6.1 Eliminating Negation and Disjunction

Our idea is to eliminate negation and disjunction symbols in the constraint set such that, if we can derive termination guarantees for the rewritten constraint set, then termination guarantees follow for the original one. This elimination process is defined via the two mappings introduced in the following definition. Let $\mathcal{T}' = \{T'_1, ..., T'_m\}$ and $\mathcal{R}' := \{R'_1, ..., R'_n\}$ be two additional sets of relational symbols such that $(\mathcal{R}' \cup \mathcal{T}') \cap (\mathcal{R} \cup \mathcal{T}) = \emptyset = \mathcal{R}' \cap \mathcal{T}'$.

DEFINITION 9. *By L_0 we denote the set of first order sentences over vocabulary $\mathcal{R} \cup \mathcal{T}$.*

- *The mapping $T_\neg : 2^{L_0} \to 2^{L_0}$ is defined as follows. For every $\Phi \subseteq L_0$ we set $T_\neg(\Phi) := \widetilde{\Phi}$, where $\widetilde{\Phi}$ equals Φ except that for every $i \in [n]$ every occurrence of $\neg \mathtt{R}_i$ ($\neg \mathtt{T}_i$) is substituted by $\mathtt{R}'_i$ ($\mathtt{T}'_i$).*
- *The mapping $T_\vee : 2^{L_0} \to 2^{L_0}$ is defined as follows. For every $\Phi \subseteq L_0$ we set $T_\vee(\Phi)$ to be $\bigcup_{\varphi \in \Phi} \widehat{\varphi}$.* □

Note that both mappings are well-defined and computable in polynomial time. We show by an example how we intend to use these mappings.

EXAMPLE 8. *Let Σ be a set of $\mathrm{TGD}^{\vee,\neg}$ and $\mathrm{EGD}^{\vee}$. If Σ has stratified witness as defined in [13], then it is easy to see that $T_\vee(T_\neg(\Sigma))$ is weakly acyclic. The converse is not true. Consider e.g. the constraint set Σ consisting of*

$$\mathtt{R}(x_1, x_2) \to \exists y \neg\, \mathtt{S}(x_1, y), \text{ and}$$
$$\mathtt{S}(x_1, x_2) \to \exists y \neg\, \mathtt{R}(x_1, y).$$ □

The example shows that, if we can derive termination guarantees for $T_\vee(T_\neg(\Sigma))$, then we are more general than stratified witness, the only termination condition known so far that covers constraints containing negation and disjunction. The next theorem fills this gap and allows to derive such termination bounds from existing termination conditions for (negation- and disjunction-free) TGDs and EGDs.

THEOREM 10. *Let Σ be a set of $\mathrm{TGD}^{\vee,\neg}$ and $\mathrm{EGD}^{\vee}$.*

- *If $T_\neg(\Sigma) \in \mathrm{CT}_{\forall\forall}$, then $\Sigma \in \mathrm{CT}_{\forall\forall}$.*
- *If $\Sigma = T_\vee(\Sigma)$ and $T_\neg(\Sigma) \in \mathrm{CT}_{\forall\exists}$, then $\Sigma \in \mathrm{CT}_{\forall\exists}$.*
- *If $T_\vee(\Sigma) \in \mathrm{CT}_{\forall\forall}$, then $\Sigma \in \mathrm{CT}_{\forall\forall}$.*
- *If $T_\vee(T_\neg(\Sigma)) \in \mathrm{CT}_{\forall\forall}$, then $\Sigma \in \mathrm{CT}_{\forall\forall}$.* □

We emphasize that, with this powerful tool at hand, we subsume both the work on chase termination in the presence of $\mathrm{EGD}^{\vee}$ from [17] and for $\mathrm{TGD}^{\vee,\neg}$ from [13].

We now turn towards the question of data complexity (i.e. in the size of the input query) of chase termination.

THEOREM 11. *Let Σ be a set of $\mathrm{TGD}^{\vee,\neg}$ and $\mathrm{EGD}^{\vee}$. If there is a function $f : \mathrm{UCQ}^{\neg} \to \mathbb{N}$ such that for every $Q \in \mathrm{UCQ}$ the depth of every chase tree for $T_\vee(T_\neg(\Sigma))$ and Q is bounded by $f(Q)$, then for every $Q' \in \mathrm{UCQ}^{\neg}$ the depth of every chase tree for Σ and Q' is also bounded by $f(Q)$.* □

In particular, it follows from this theorem that if we can data-independently derive polynomial data complexity for $T_\vee(T_\neg(\Sigma))$, which is the case for all termination conditions for the chase that have been developed so far, then the depth of every chase tree for Σ is also polynomial.

6.2 Sequence-Dependent Termination

As a starting point for our work, we make the observation that the stratification condition introduced in [14] does not belong to $\mathrm{CT}_{\forall\forall}$, as stated by the authors of [14], but to $\mathrm{CT}_{\forall\exists}$. It is the cornerstone for all further sequence-dependent termination conditions that we develop here. We start with an introduction to stratified constraint sets.

6.2.1 Stratification

The main idea behind stratification is to decompose the constraint set into independent subsets that are then separately tested for weak acyclicity [17]. More precisely, the decomposition splits the input constraint set into subsets of constraints that may cyclically cause to fire each other. The idea is that the termination guarantee for the whole constraint set should follow if weak acyclicity holds for each subset in the decomposition. The basis for such a decomposition is the binary relation $\prec$ on the constraint set.

DEFINITION 10. *(see [14]) Given two TGDs or EGDs $\alpha, \beta \in \Sigma$ we define $\alpha \prec \beta$ iff there exists a relational database instance I and $\overline{a}, \overline{b}$ such that (i) $I \not\models \alpha(\overline{a})$, (ii) $I \models \beta(\overline{b})$, (iii) $I \overset{\alpha,\overline{a}}{\to} J$, and (iv) $J \not\models \beta(\overline{b})$.* □

The actual definition of stratification then relies, as outlined before, on the notion of weak acyclicity.

DEFINITION 11. *(see [14]) The chase graph $G(\Sigma) = (\Sigma, E)$ of a set of constraints Σ contains a directed edge (α, β) between two constraints iff $\alpha \prec \beta$. We call Σ stratified iff the constraints in every cycle of $G(\Sigma)$ are weakly acyclic.* □

Stratification strictly generalizes weak acyclicity [14] in the sense that (i) if Σ is weakly acyclic, then it is also stratified, and (ii) there are constraint sets that are stratified but not weakly acyclic. The authors of [14] state the following:

Let Σ be a fixed set of stratified constraints. Then, there exists a polynomial $Q \in \mathbb{N}[X]$ such that for every database instance I, the length of every chase sequence is bounded by $Q(|dom(I)|)$. Thus, $\Sigma \in \mathrm{CT}_{\forall\forall}$. □

Unfortunately, as we show in the following example, this statement is not true:

EXAMPLE 9. *Let $\Sigma := \{\alpha_1, ..., \alpha_4\}$, where*

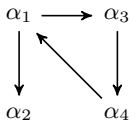


Figure 1: Chase graph for Example 9.

$\alpha_1 := R(x_1) \rightarrow S(x_1, x_1)$,
$\alpha_2 := S(x_1, x_2) \rightarrow \exists z T(x_2, z)$,
$\alpha_3 := S(x_1, x_2) \rightarrow T(x_1, x_2), T(x_2, x_1)$, *and*
$\alpha_4 := T(x_1, x_2), T(x_1, x_3), T(x_3, x_1) \rightarrow R(x_2)$.

We now give an instance for which the chase does not necessarily terminate. Consider the database $\{R(a)\}$ *and the chase sequence which applies the constraints in the order* $\alpha_1, ..., \alpha_4, \alpha_1, ..., \alpha_4, ...$ *and so on. The first steps of the resulting chase sequence look as follows:*

$\{R(a)\}$
$\xrightarrow{\alpha_1,a}$ $\{R(a), S(a,a)\}$
$\xrightarrow{\alpha_2,a,a}$ $\{R(a), S(a,a), T(a,n_1)\}$
$\xrightarrow{\alpha_3,a,a}$ $\{R(a), S(a,a), T(a,n_1), T(a,a)\}$
$\xrightarrow{\alpha_4,a,n_1,a}$ $\{R(a), S(a,a), T(a,n_1), T(a,a), R(n_1)\}$
$\xrightarrow{\alpha_1,n_1}$ $\{R(a), S(a,a), T(a,n_1), T(a,a), R(n_1), S(n_1,n_1)\}$
$\xrightarrow{\alpha_2,n_1,n_1}$ $\{R(a), S(a,a), T(a,n_1), T(a,a), R(n_1), S(n_1,n_1), T(n_1,n_2)\}$
$\xrightarrow{\alpha_3,n_1,n_1}$ $\{R(a), S(a,a), T(a,n_1), T(a,a), R(n_1), S(n_1,n_1), T(n_1,n_2), T(n_1,n_1)\}$
$\xrightarrow{\alpha_4,n_1,n_2,n_1}$ $\{R(a), S(a,a), T(a,n_1), T(a,a), R(n_1), S(n_1,n_1), T(n_1,n_2), T(n_1,n_1), R(n_2)\}$,
$\xrightarrow{\alpha_1,n_2}$ ...

where n_1, n_2 *are fresh null values. It can be easily seen that this sequence is infinite. The chase graph for* Σ *is depicted in Figure 1. The only cycle in it contains full* TGDs *only and thus is weakly acyclic. It follows that* Σ *is stratified.* □

As a consequence, unlike weak acyclicity, stratification does not ensure termination in the sense of $\mathrm{CT}_{\forall\forall}$ as claimed in [14]. However, we can prove another, equally useful result for the stratification class. If a set of constraints is stratified, we cannot ensure termination in the sense of $\mathrm{CT}_{\forall\forall}$, but $\mathrm{CT}_{\forall\exists}$, as stated in the following.

THEOREM 12. *Let* Σ *be stratified. Then, there exists a polynomial* $Q \in \mathbb{N}[X]$ *such that for any database instance* I *there is a terminating chase sequence whose length is bounded by* $Q(|dom(I)|)$. *Thus,* $\Sigma \in \mathrm{CT}_{\forall\exists}$. □

How can we use this result in practice? The first idea is to apply the chase in a breadth-first manner, i.e. generating a tree whose root is the start instance, its children are obtained by applying one chase step on the start instance and the tree is expanded in breadth-first manner. As it turns out, we are in a much better situation here. We can use the chase graph to statically construct the order in which the constraints can be applied to ensure a terminating chase sequence.

THEOREM 13. *Let* Σ *be a fixed set of constraints. If for every strongly-connected component* Σ' *of* $G(\Sigma)$ *it holds that* $\Sigma' \in \mathrm{CT}_{\forall\forall}$, *then* $\Sigma \in \mathrm{CT}_{\forall\exists}$ *and a terminating chase sequence can be statically constructed.* □

The idea of the proof is to impose a partial order on the strongly connected component of $G(\Sigma)$ with the help of $\prec$. This partial order is extended to a total order via topological sorting. Then, we can chase with each strongly connected components of $G(\Sigma)$ individually. Hence, we avoid the overhead of branching in the breadth-first chase and therefore reduce the complexity of generating a chase result.

6.2.2 The ∀∃-T-Hierarchy

We continue our study of $\mathrm{CT}_{\forall\exists}$ by combining our ideas related to the T-hierarchy which we introduced in [31] with stratification. In particular, we use the T-hierarchy as a tool to define the ∀∃-T-hierarchy, which covers larger fragments of $\mathrm{CT}_{\forall\exists}$ than stratification. We refer the reader to [32] for a formal definition of the T-hierarchy.

DEFINITION 12. *Let* $k \geq 2$. *We say* Σ *is in* $\forall\exists$-$T[k]$ *iff every cycle of* $G(\Sigma)$ *is in* $T[k]$. □

EXAMPLE 10. *The constraint set* $\Sigma_k := \{\alpha_k\}$, *where* $\alpha_k := S(x_k), R_k(x_1, ..., x_k) \rightarrow \exists y R_k(y, x_1, ..., x_{k-1})$ *is in* $T[k+1] \setminus T[k]$ *(cf. [31]). We also have that* $\alpha_k \prec \alpha_k$, *so we can conclude that* $\Sigma_k \in \forall\exists\text{-}T[k+1] \setminus \forall\exists\text{-}T[k]$. □

As a corollary to Theorem 13 we can state that for every database instance a terminating chase sequence can always be statically constructed from the underlying chase graph.

COROLLARY 4. *Let* $k \geq 2$ *and* Σ *be a fixed set of constraints in* $\forall\exists$-$T[k]$ *constraints. For every database instance* I *a terminating chase for* I *and* Σ *sequence with length polynomial in* $|dom(I)|$ *can be statically constructed.* □

The corollary's use is illustrated by the next example.

EXAMPLE 11. *Let* Σ *be the following set of* TGDs*:*

$\alpha_1 := R(x_1) \rightarrow S(x_1, x_1)$,
$\alpha_2 := S(x_1, x_2) \rightarrow \exists z T(x_2, z)$,
$\alpha_3 := S(x_1, x_2) \rightarrow T(x_1, x_2), T(x_2, x_1)$,
$\alpha_4 := T(x_1, x_2), T(x_1, x_3), T(x_3, x_1) \rightarrow R(x_2)$,
$\alpha_5 := T(x_1, x_2) \rightarrow E(x_1, x_2)$,
$\alpha_6 := U(x_1), E(x_1, x_2) \rightarrow E(x_2, x_1)$ *and*
$\alpha_7 := U(x_1), E(x_1, x_2) \rightarrow \exists y E(x_2, y), E(y, x_1)$.

It is not hard to see that $\Sigma \in \mathrm{CT}_{\forall\exists}$ *because we can first apply a terminating chase sequence to satisfy* $\{\alpha_1, ..., \alpha_4\}$ *and then apply the rest of the constraints afterward. No termination condition for the chase introduced in previous work recognizes that there is always a terminating chase sequence for* Σ*:* Σ *is not in any level of the T-hierarchy because for* $\{\alpha_1, ..., \alpha_4\}$ *there may be non-terminating chase sequences and* Σ *can also not be stratified because* $\{\alpha_6, \alpha_7\}$ *is not weakly acyclic. We show that* $\Sigma \in \forall\exists$-$T[2]$. *The chase graph is depicted in Figure 2. There are two strongly connected components, namely* $\{\alpha_1, \alpha_3, \alpha_4\}$ *and* $\{\alpha_6, \alpha_7\}$. *As both are in* $T[2]$, *we can conclude that* $\Sigma \in \forall\exists$-$T[2]$. *We can also determine such a sequence statically from the shape of the chase graph. We chase sequentially with (i)* $\{\alpha_1, \alpha_3, \alpha_4\}$ *until they are satisfied, (ii) then take* α_2 *and apply it until it is satisfied, (iii) afterwards we perform every possible chase step with* α_5, *and (iv) finally chase with the second strongly connected component* $\{\alpha_6, \alpha_7\}$. *At the end, all constraints are satisfied and therefore the chase sequence is finite.* □

This example shows us that a further decomposition of the strongly connected components of the chase graph via the T-hierarchy can lead to an improved chase termination condition. And indeed it can be shown that ∀∃-T[k] is a proper extension of $T[k]$ for every $k \geq 2$:

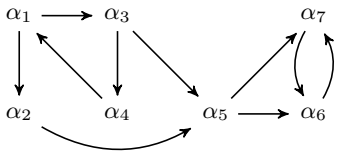


Figure 2: Chase graph for Example 11.

PROPOSITION 3. *If $k \geq 2$, then $T[k] \subset \forall\exists\text{-}T[k]$.* □

It was already shown in Example 11 that the extension is proper. The containment follows from the observation that if $\Sigma \in T[k]$, then so is every subset of Σ.

7. CONCLUSION

In previous investigations semantic query optimization and optimizations based on complex type hierarchies have been studied separately although both topics have striking commonalities. Unifying these two research areas, we have developed a logical framework that seamlessly integrates both techniques and, in the general case, provides better optimization results than their application in two isolated, subsequent stages. We also provided algorithms to enumerate optimized queries as well as results on the complexity of related decision and satisfiability problems. The applicability of our method depends on chase termination, and in response we proposed novel termination conditions in the presence of negation and disjunction, thereby opening up the field of sequence-dependent chase termination conditions.

In future work, we will investigate extending the use of negation. More research is required in identifying larger decidable fragments of $\mathrm{CT}_{\forall\forall}$ and $\mathrm{CT}_{\forall\exists}$. On the practical side we plan to implement and evaluate our approach.

8. REFERENCES

[1] A. Deutsch, L. Popa and V. Tannen. Query Reformulation with Constraints. *SIGMOD Record*, 35(1):65–73, 2006.

[2] S. Abiteboul, G. Lausen, H. Uphoff, and E. Waller. Methods and rules. In *SIGMOD*, pages 32–41, New York, NY, USA, 1993. ACM.

[3] C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *J. ACM*, 31(4):718–741, 1984.

[4] M. Bruynooghe, J. Gallagher, and W. Van Humbeeck. Inference of Well-Typings for Logic Programs with Application to Termination Analysis. In *SAS*, pages 35–51, 2005.

[5] P. Buneman, W. Fan, and S. Weinstein. Interaction between path and type constraints. *ACM Trans. Comput. Logic*, 4(4):530–577, 2003.

[6] A. Calì, G. Gottlob, and T. Lukasiewicz. A General Datalog-based Framework for Tractable Query Answering over Ontologies. In *PODS*, 2009.

[7] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-lite Family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

[8] D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive Query Containment and Answering under Description Logic Constraints. *ACM Trans. Comput. Log.*, 9(3), 2008.

[9] U. Chakravarthy, J. Grant, and J. Minker. Logic-based Approach to Semantic Query Optimization. *TODS*, 15(2):162–207, 1990.

[10] E. Chan. Containment and Minimization of Positive Conjunctive Queries in OODB's. In *PODS*, 1992.

[11] D. Maier, A. Mendelzon and Y. Sagiv. Testing Implications of Data Dependencies. In *SIGMOD*, pages 152–152, 1979.

[12] O. de Moor, D. Sereni, P. Avgustinov, and M. Verbaere. Type Inference for Datalog and its Application to Query Optimisation. In *PODS*, pages 291–300, 2008.

[13] A. Deutsch, B. Ludäscher, and A. Nash. Rewriting Queries Using Views with Access Patterns under Integrity Constraints. *Theor. Comput. Sci.*, 371(3):200–226, 2007.

[14] A. Deutsch, A. Nash, and J. Remmel. The Chase Revisited. In *PODS*, pages 149–158, 2008.

[15] G. Dong and J. Su. Conjunctive Query Containment with Respect to Views and Constraints. *Inf. Process. Lett.*, 57(2):95–102, 1996.

[16] R. Fagin. Horn Clauses and Database Dependencies. *J. ACM*, 29(4):952–985, 1982.

[17] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[18] W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM*, 49(3):368–406, 2002.

[19] T. Frühwirth, E. Shapiro, M. Vardi, and E. Yardeni. Logic Programs as Types for Logic Programs. In *LICS*, pages 300–309, 1991.

[20] A. Fuxman, P. Kolaitis, R. Miller, and W.-C. Tan. Peer Data Exchange. *TODS*, 31(4):1454–1498, 2006.

[21] J. Gallagher and G. Puebla. Abstract Interpretation over Non-Deterministic Finite Tree Automata for Set-Based Analysis of Logic Programs. In *PADL*, pages 243–261, 2002.

[22] A. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, 10(4):270–294, 2001.

[23] J. Henriksson and J. Maluszynski. Static Type-Checking of Datalog with Ontologies. In *PPSWR*, pages 76–89, 2004.

[24] D. Johnson and A. Klug. Testing Containment of Conjunctive Queries Under Functional and Inclusion Dependencies. In *PODS*, pages 164–169, 1982.

[25] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *J. ACM*, 42(4):741–843, 1995.

[26] P. Kolaitis, D. Martin, and M. Thakur. On the Complexity of the Containment Problem for Conjunctive Queries with Built-in Predicates. In *PODS*, pages 197–204, 1998.

[27] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[28] A. Levy and D. Suciu. Deciding Containment for Queries with Complex Objects. In *PODS*, pages 20–31, 1997.

[29] W. Litwin and T. Risch. Main Memory Oriented Optimization of OO Queries Using Typed Datalog with Foreign Predicates. *TKDE*, 4(6):517–528, 1992.

[30] B. Marnette. Generalized Schema-Mappings: From Termination to Tractability. In *PODS*, pages 13–22, 2009.

[31] M. Meier, M. Schmidt, and G. Lausen. On Chase Termination Beyond Stratification. *PVLDB*, 2(1), 2009.

[32] M. Meier, M. Schmidt, and G. Lausen. On Chase Termination Beyond Stratification. *Technical Report*, 2009.

[33] T. Milo and D. Suciu. Type Inference for Queries on Semistructured Data. In *PODS*, pages 215–226, 1999.

[34] D. Olteanu, J. Huang, and C. Koch. SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases. In *ICDE*, pages 640–651, 2009.

[35] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *ICDE*, pages 251–260, 1995.

[36] L. Popa, A. Deutsch, A. Sahuguet, and V. Tannen. A Chase Too Far? In *SIGMOD*, pages 273–284, 2000.

[37] L. Popa and V. Tannen. An Equational Chase for Path-Conjunctive Queries, Constraints, and Views. In *ICDT*, pages 39–57, 1999.

[38] M. Schäfer and O. de Moor. Type inference for datalog with complex type hierarchies. In *POPL*, pages 145–156, 2010.

[39] M. Schmidt, M. Meier, and G. Lausen. Foundations of SPARQL Query Optimization. In *ICDT*, 2010.

[40] L. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3:1–22, 1976.

[41] D. Toman and G. E. Weddell. On Path-functional Dependencies as First-class Citizens in Description Logics. In *Description Logics*, 2005.

[42] D. Zook, E. Pasalic, and B. Sarna-Starosta. Typed Datalog. In *PADL*, pages 162–182, 2009.

APPENDIX

A. SELECTED PROOFS

Proof of Lemma 3. Without loss of generality assume that there is some instance I such that $I \models \mathcal{S}$ and $Q(I) \neq \emptyset$. Otherwise, the claim is trivial.

The statement of this corollary can be reduced to Theorem 9 in [13]. It states that for $W, W' \in \text{UCQ}^\neg$ we have that $W \sqsubseteq_\Sigma W'$ iff $W^{\Sigma \cup \Sigma_\neg} \sqsubseteq W'$.

Our reduction is as follows. For every constant c in Q' and in $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$ and for every $R \in \mathcal{R} \cup \mathcal{T}$ we add the constraint

$$R(c, x_2, ..., x_{ar(\mathtt{R})}) \rightarrow \bigwedge_{l \in \text{TYPE}(c)} l$$
$$R(x_1, c, x_3, ..., x_{ar(\mathtt{R})}) \rightarrow \bigwedge_{l \in \text{TYPE}(c)} l$$
$$\dots$$
$$R(x_1, , ..., x_{ar(\mathtt{R})-1}, c) \rightarrow \bigwedge_{l \in \text{TYPE}(c)} l$$

to $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$ and call the resulting constraint set Σ_{new}. We conclude that

- for all database instances I it holds that $I \models \mathcal{S} \cup \Sigma_\neg \iff I \models \Sigma_{new}$, and
- $Q'^{\mathcal{S} \cup \Sigma_\neg}$ is homomorphically equivalent to $Q'^{\Sigma_{new}}$.

The first bullet is obvious. For the second bullet note that $Q'^{\mathcal{S}}$ can be viewed as $Q'^{\Sigma_{new}}$ obtained via a certain chase order. Then, the statement can be obtained from Theorem 9 in [13].

Proof of Theorem 2. Proof of the first equivalence: for the forward direction assume that $Q' \sqsubseteq_\Sigma Q$ holds. Let $comp(Q')^\Sigma = \bigvee_{i \in [k]} P_i$ and $comp(P_i) = \bigvee_{j \in [l_i]} P_{i,j}$. Note that for every P_i, every $R \in \mathcal{R} \cup \mathcal{T}$ and every tuple $\overline{x}$ of constants in $comp(Q')^\Sigma$ and variables in $ans(P_i)$ we have that either $R(\overline{x})$ or $\neg R(\overline{x})$ is in the body of P_i. We have that $ans(P_i) \in P_i(P_i)$, which implies $ans(P_i) \in Q'^\Sigma(P_i)$. As $db(P_i) \models \Sigma$, it follows that $ans(P_i) \in Q'(P_i)$. Using the assumption, we obtain $ans(P_i) \in Q(P_i)$. Assume $Q = \bigvee_{j \in [l]} P_j$. For some $j \in [l]$ we have that $ans(P_i) \in Q_j(P_i)$. We will show that for all $j' \in [l_i]$ it holds that $ans(P_i) = ans(P_{i,j'}) \in Q_j(P_{i,j'})$. Suppose not. Then, there is $j'' \in [l_i]$ such that $ans(P_i) = ans(P_{i,j''}) \notin Q_j(P_{i,j''})$. Then, for every homomorphism h that maps Q_j^+ to $P_{i,j''}$ there is some atom A such that $\neg A \in Q_j^-$ and $h(A) \in P_{i,j''}$. Let μ be a homomorphism that witnesses $ans(P_i) \in Q_j(P_i)$. Let A be an atom such that $\neg A \in Q_j^-$. We can conclude that $\neg \mu(A) \in P_i^-$ because $Q \in \text{UCQ}^{\neg/2}$. Therefore, μ witnesses $ans(P_i) = ans(P_{i,j''}) \in Q_j(P_{i,j''})$, which is a contradiction. We have shown that for all $j' \in [l_i]$ it holds that $Q_j \rightarrow P_{i,j'}$, which implies our claim. For the backward direction assume that for all $P' \in comp(Q')^\Sigma$ there exists $P \in Q$ such that it holds that $P \rightarrow P'$. Hence, $comp(Q')^\Sigma \sqsubseteq Q$. Observe that $Q' \equiv comp(Q') \equiv_\Sigma comp(Q')^\Sigma \sqsubseteq Q$ implies the claim.

The second equivalence is a corollary from the first one.

Proof of Corollary 2. The statement of this corollary can be reduced to Theorem 2 as follows. For every constant c in Q' and in $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$ and for every $R \in \mathcal{R} \cup \mathcal{T}$ we add the constraint

$$R(c, x_2, ..., x_{ar(\mathtt{R})}) \rightarrow \bigwedge_{l \in \text{TYPE}(c)} l$$
$$R(x_1, c, x_3, ..., x_{ar(\mathtt{R})}) \rightarrow \bigwedge_{l \in \text{TYPE}(c)} l$$
$$\dots$$
$$R(x_1, , ..., x_{ar(\mathtt{R})-1}, c) \rightarrow \bigwedge_{l \in \text{TYPE}(c)} l$$

to $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$ and call the resulting constraint set Σ_{new}. We conclude that

- for all database instances I it holds that $I \models \mathcal{S} \iff I \models \Sigma_{new}$, and
- $Q'^{\mathcal{S}}$ is homomorphically equivalent to $Q'^{\Sigma_{new}}$.

The first bullet is obvious. For the second bullet note that $Q'^{\mathcal{S}}$ can be viewed as $Q'^{\Sigma_{new}}$ obtained via a certain chase order. Then, the corollary can be obtained from Theorem 2.

Proof of Theorem 3. Without loss of generality, we assume that $H(\mathcal{S})$ contains for every $T \in \mathcal{T}$ some $\overline{T} \in \mathcal{T}$ such that $\overline{T}$ reflects the complement of T, i.e. we have constraints (1) $\neg T(x) \rightarrow \overline{T}(x)$, (2) $\neg \overline{T}(x) \rightarrow T(x)$, (3) $T(x) \rightarrow \neg \overline{T}(x)$, (4) $\overline{T}(x) \rightarrow \neg T(x)$, and (5) $T(x), \overline{T}(x) \rightarrow \neg T(x)$.

We define a mapping $E : \text{UCQ}^\neg \rightarrow \text{UCQ}^\neg$ that substitutes all negative $\mathcal{T}$-literals $\neg T(...)$ by $\overline{T}(...)$. This operation preserves equivalence under $\mathcal{S}$.

Let $Q \in \text{UCQ}^{\neg/2}$ such that $Q \mapsto Q^\mathcal{S}$ is computable. The algorithm is as follows:

1. Initialize $M := \emptyset$.
2. Compute $comp(Q)^\mathcal{S}$.
3. Enumerate all $Q' \in \text{UCQ}^{\neg/2}$ with $c(Q') \leq c(Q)$:
   (a) Rewrite Q' to Q'' such that $Q' \equiv Q''$ and every $q \in Q''$ contains all constants from Q, Q'' and $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$
   (b) Compute $comp(Q'')^\mathcal{S}$.
   (c) Test whether $E(Q'') \equiv_\mathcal{S} E(Q)$ holds with the help of Corollary 2 and if so, add Q' to M.
4. Output $\{Q'' \in M \mid \text{for all } Q''' \in M \text{ it holds that } c(Q'') \leq c(Q''')\}$.

By Proposition 1 this algorithm terminates. It follows from Corollary 2 that it is sound and complete for finding $(\text{UCQ}^{\neg/2}, c, \mathcal{S})$-minimal rewritings of the input query.

Proof of Lemma 4.

- We have that $\Sigma(\mathcal{S})|_\mathcal{R} \subseteq \text{CT}_{\forall\forall} \implies \Sigma(\mathcal{S})|_\mathcal{R} \cup H(\mathcal{S}) \subseteq \text{CT}_{\forall\forall} \implies \Sigma(\mathcal{S}) \cup H(\mathcal{S}) \subseteq \text{CT}_{\forall\forall}$ because every variable that occurs in some $\mathcal{T}$-literal also occurs in some $\mathcal{R}$-atom. Therefore, we can conclude that for every $\overline{Q} \in \text{UCQ}^\neg$ the mapping $Q \mapsto Q^\mathcal{S}$ is computable.
- Let $\Sigma(\mathcal{S}) = \Sigma(\mathcal{S})|_\mathcal{R} \subseteq \text{CT}_{\forall\exists}$. We can conclude that $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \text{CT}_{\forall\exists}$ because we can first chase with $\Sigma(\mathcal{S})$ according to a strategy that guarantees termination and afterward apply the constraints in $H(\mathcal{S})$. Therefore, we can conclude that for every $\overline{Q} \in \text{UCQ}^\neg$ the mapping $Q \mapsto Q^\mathcal{S}$ is computable.

Proof of Theorem 4. Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \text{CT}_{\forall\exists}$. We show that $\mathcal{S}$ is satisfiable iff $(\bigvee_{\mathtt{R} \in \mathcal{R}} q() \leftarrow R(x_1, ..., x_{ar(\mathtt{R})}))^\mathcal{S} \neq \text{FALSE}$. The backward direction is standard. For the forward direction assume that $\mathcal{S}$ is satisfiable but $(\bigvee_{\mathtt{R} \in \mathcal{R}} q() \leftarrow R(x_1, ..., x_{ar(\mathtt{R})}))^\mathcal{S} = \text{FALSE}$. Let I be a finite database instance such that $I \models \mathcal{S}$ and $I|_\mathcal{R} \neq \emptyset$. It follows from the proof of Lemma 1 that for every intermediate (during the application of the chase) union of queries Q there is $q \in Q$ such that $db(q) \rightarrow I$. We can conclude that $(\bigvee_{\mathtt{R} \in \mathcal{R}} q() \leftarrow R(x_1, ..., x_{ar(\mathtt{R})}))^\mathcal{S} \neq \text{FALSE}$, which is a contradiction.

Proof of Theorem 5. We prove NP-hardness by reduction from CNF-SAT, the satisfiability problem for propositional formulas in conjunctive normal form.

Let $\alpha := \bigwedge_{i\in[m]} B_i$ be given, where $B_i = x_{i,1} \vee \ldots \vee x_{i,k_i} \vee \neg y_{i,1} \vee \ldots \vee \neg y_{i,l_i}$ and the set of propositional variables that occurs in α is $\{x_1, \ldots, x_n\}$.

We encode α in a typed relational schema as follows. Note that we denote strings by surrounding quotation marks.

$$\exists x_1, \ldots, x_n (\bigwedge_{i\in[m]} ((\bigwedge_{j\in[k_i]} R(\text{`}i\text{'}, \text{`}j\text{'}, \text{`}+\text{'}, \text{`}x_{i,j}\text{'}, x_{i,j})) \wedge (\bigwedge_{j\in[l_i]} R(\text{`}i\text{'}, \text{`}j\text{'}, \text{`}\neg\text{'}, \text{`}y_{i,j}\text{'}, y_{i,j})))),$$

$$(\bigwedge_{j\in[k_i]} R(\text{`}i\text{'}, \text{`}j\text{'}, \text{`}+\text{'}, \text{`}x_{i,j}\text{'}, x_{i,j})) \wedge (\bigwedge_{j\in[l_i]} R(\text{`}i\text{'}, \text{`}j\text{'}, \text{`}\neg\text{'}, \text{`}y_{i,j}\text{'}, y_{i,j})) \rightarrow \bigvee_{j\in[k_i]} L_1(x_{i,j}) \vee \bigvee_{j\in[l_i]} L_0(y_{i,j}), \text{ for all } i \in [m],$$

$$R(x_1, x_2, x_3, x_4, x_5) \rightarrow L_0(x_5) \vee L_1(x_5),$$
$$L_0(x) \rightarrow \neg L_1(x), \text{ and}$$
$$L_1(x) \rightarrow \neg L_0(x).$$

The last two constraints constitute the type hierarchy $H(\mathcal{S})$ and all other constraints constitute $\Sigma(\mathcal{S})$, which means that the constraints obey the syntactic restrictions in order to form valid typed relational schema. It is standard to verify that α is satisfiable iff $\mathcal{S}$ is satisfiable.

Proof of Theorem 6. We give a Σ_2^P-algorithm that tests satisfiability, i.e. an NP algorithm with coNP oracle. Initially, guess some predicate $R \in \mathcal{R}$ and consider the query $q() \leftarrow R(x_1, \ldots, x_{ar(R)})$. Then compute $q^{\mathcal{S}}$, using the following modification of the algorithm from Definition 6. In each chase step starting with query q_i as input, guess some $\alpha \in \Sigma(\mathcal{S}) \cup H(\mathcal{S})$ and constants $\overline{a}$ for the universally quantified variables in α such that $db(q_i) \models body(\alpha(\overline{a}))$ (testing the latter condition can be done in polynomial time). Then use the coNP oracle to verify that there are no constants $\overline{b}$ for the existentially quantified variables in the head of α such that $db(q_i) \models head(\alpha(\overline{a}, \overline{b}))$. If α is a TGD or EGD with disjunction in the head, i.e. is of the form $\alpha := \phi \rightarrow \psi_1 \vee \cdots \vee \psi_n$, guess $i \in [n]$ and perform a chase step with $\alpha' := \phi \rightarrow \psi_i$ instead of α. Otherwise, if the head of constraint α is disjunction-free, simply perform a chase step with α.

By assumption, the above algorithm terminates in k steps, where k is polynomially bounded by the depth of the chase tree, so it is easy to see that the algorithm is in Σ_2^P. It is standard to show that $\mathcal{S}$ is satisfiable iff $q_{k+1} \neq$ FALSE (in case $q_i =$ FALSE for some $i < k+1$, we set $q_{k+1} :=$ FALSE).

Proof of Theorem 7.

We prove Π_2^P-hardness by a reduction from the containment problem for conjunctive queries with inequalities. In [26] it was shown that this problem is Π_2^P-complete even for boolean queries. Thus, we can restrict ourselves to boolean queries which we represent as the set of atoms in the query's body. Without loss of generality, we can assume that q_1 and q_2 have disjoint sets of variables when we want to test whether $q_1 \sqsubseteq q_2$ holds.

In the first step, we reduce the containment problem for such queries to the equivalence problem via the reduction $q_1 \sqsubseteq q_2 \iff q_1 \equiv q_1 \cup q_2$.

In the second step we reduce the equivalence problem for conjunctive queries with inequalities to REWRITE$(\text{CQ}, c, \mathcal{S})$ for the fixed typed relational schema $\mathcal{S}$ with

$$\Sigma(\mathcal{S}) = \{I(x,x) \rightarrow T(x)\} \cup \{J(x,y) \rightarrow x = y\} \cup$$
$$\{I(x,y), J(x,y) \rightarrow T(x)\} \cup$$
$$\{R(x_1, ..., x_{ar(R)}) \rightarrow D(x_1), ..., D(x_{ar(R)}) | R \in \mathcal{R}\} \cup$$
$$\{D(x_1), D(x_2) \rightarrow I(x_1, x_2) \vee J(x_1, x_2)\}, \text{ and}$$
$$H(\mathcal{S}) = \{T(x) \rightarrow \neg T(x)\},$$

where I, J, D, T are fresh relational symbols. Intuitively, J simulates the equality predicate and I its complement with respect to the active domain. Given two conjunctive queries with inequalities q_1, q_2 as input, we translate to q_1' and q_2' by substituting every inequality atom of the form $x_i \neq x_k$ by $I(x_i, x_k)$. We can conclude that $q_1 \equiv q_2 \iff q_1' \equiv_{\mathcal{S}} q_2'$ holds, which finishes this proof.

Proof of Theorem 8. Given Theorem 7, it suffices to prove membership in Π_2^P. We give an algorithm which is appropriate for our needs. Given (Q, Q') as input our test essentially consists of an application of Theorem 2 and Corollary 2. W.l.o.g. every $q \in Q'$ contains all constants from Q, Q' and Σ. We test whether for all $P' \in comp(Q'^{\Sigma})$ there is $P \in Q$ such that for all $P'' \in comp(P')$ it holds that $P \rightarrow P''$ and whether for all $P \in comp(Q^{\Sigma})$ there is $P' \in Q'$ such that for all $P'' \in comp(P)$ it holds that $P' \rightarrow P''$. This is doable in Π_2^P-time.

Proof of Theorem 9. We show that the complement of MINIMAL$(\text{UCQ}, c, \mathcal{S})$ is in Σ_3^P. In order to see this we can use the following algorithm. Given Q as input, we guess Q' such that $c(Q') < c(Q)$ and test whether $(Q, Q') \in$ REWRITE$(\text{UCQ}, c, \mathcal{S})$ (see Theorem 8) holds. Clearly, all this is doable in Σ_3^P-time .

Proof of Theorem 10.

- It is immediate from the definition of chase steps that the set of chase sequences with $T_{\neg}(\Sigma)$ coincide with the chase steps of Σ up to renaming of negative literals with $T_{\neg}$ and unsatisfiability.
- Let Σ contain no disjunctions. Observe again that the set of chase sequences with $T_{\neg}(\Sigma)$ coincide with the chase steps of Σ up to renaming of negative literals with $T_{\neg}$ and unsatisfiability.
- Suppose not. Then, there is an infinite chase sequence $Q_0 \xrightarrow{\alpha_1, \overline{a}_1} Q_1 \xrightarrow{\alpha_2, \overline{a}_2} Q_2 \ldots$, where $\{\alpha_i | i \in \mathbb{N}\} \subseteq \Sigma$. It follows that there is $q_0 \in Q_0 \cap \text{CQ}^{\neg}$ and for every $i \in \mathbb{N}$ there is $q_i \in \text{CQ}^{\neg} \cap Q_i$ and $\overline{\alpha}_i \in \widehat{\alpha}_i$ such that $q_0 \xrightarrow{\overline{\alpha}_1, \overline{a}_1} q_1 \xrightarrow{\overline{\alpha}_2, \overline{a}_2} q_2 \ldots$ is an infinite chase sequence, which implies $T_{\vee}(\Sigma) \notin \text{CT}_{\forall\forall}$.
- Follows from bullet one and three.

Proof of Theorem 13. Let the chase graph $G(\Sigma) = (\Sigma, E)$ be given. We write $\alpha \sim \beta$ if and only if α and β are contained in a common cycle in $G(\Sigma)$ or $\alpha = \beta$. Note that $\sim$ is an equivalence relation.

Let $\Sigma/\sim = \{W_1, ..., W_n\}$ and $E' := \{(W_i, W_j) | i, j \in [n], i \neq j,$ there is some $\alpha_i \in W_i, \beta_j \in W_j$ such that $\alpha_i \prec \beta_j\}$. Let $W_1', ..., W_n'$ be a topological sorting of $(\Sigma/\sim, E')$. Note that $W_1', ..., W_n'$ are the strongly connected components of the chase graph and constraint sets that are not involved in any cycle in the chase graph, therefore the chase terminates independently of the database instance and independently of the chase order for these constraint sets. Let I_0 be an arbitrary database instance. Let I_i be obtained from I_{i-1} by chasing with W_i' according to the chase strategy fro the prerequisites for every $i \in [n]$. It holds that $I_2 \models W_1'$. Otherwise there is some $\alpha \in W_1'$, $\beta \in W_2'$ and a database instance I such that $I \models \alpha$, but $I \xrightarrow{\beta, \overline{b}} J \not\models \alpha$. But this implies $\beta \prec \alpha$ which means W_2' must come before W_1' in the topological sorting of $(\Sigma/\sim, E')$. Using induction on n it can be seen that $I_n \models \Sigma$. Observe that $W_1', ..., W_n'$ is a partition of Σ.