A GENERAL DATALOG-BASED FRAMEWORK FOR TRACTABLE QUERY ANSWERING OVER ONTOLOGIES 2 h 20 24/7

Abstract. Ontologies and rules play a central role in the development of the Semantic Web. Recent research in this context focuses especially on highly scalable formalisms for the Web of Data, which may highly benefit from exploiting database technologies. In this paper, as a first step towards closing the gap between the semantic Web and databases, we introduce a family of expressive extensions of Datalog, called Datalog±, as a new paradigm for query answering over ontologies. The Datalog± family admits existentially quantified variables in rule heads, and has suitable restrictions to ensure highly efficient ontology querying. We show in particular that Datalog± encompasses and generalizes the tractable description logic EL and the DL-Lite family of tractable description logics, which are the most common tractable ontology languages in the context of the semantic Web and databases. We also show how stratified negation can be added to Datalog± while keeping ontology querying tractable. Furthermore, the Datalog± family is of interest in its own right, and can, moreover, be used in various contexts such as data integration and data exchange. it paves the way for applying results from databases to the context of the semantic Web.

Contents

MÔ HÌNH DATALOG TỔNG QUÁT ĐỂ TRẢ LỜI TRUY VẤN TRACTABLE TRÊN CÁC BẢN THỂ HỌC

Tóm tắt. Bản thể học và luật đóng vai trò quan trọng trong quá trình xây dựng Web Ngữ Nghĩa. Gần đây, trong lĩnh vực này, các nhà nghiên cứu tập trung nhiều vào các mô hình có khả năng mở rộng cho Web dữ liệu, thông qua việc khai thác các công nghệ cơ sở dữ liệu. Trong bài báo này, với tư các là bước khởi đầu trong quá trình hướng đến thu hẹp khoảng cách giữa Web ngữ nghĩa và cơ sở dữ liệu, chúng tôi trình bày một họ ngôn ngữ Datalog mở rộng giàu khả năng diễn đạt, được gọi là Datalog ±, đây được xem là một mô hình mới để trả lời truy vấn trên các bản thể học. Họ Datalog ± thừa nhận các biến lượng từ tồn tại trong phần đầu của luật, và có những ràng buộc thích hợp đảm bảo truy vấn bản thể học đạt hiệu quả cao. Đặc biệt, chúng ta sẽ thấy Datalog ± bao hàm và khái quát hóa logic mô tả dễ kiểm soát EL và họ logic mô tả dễ kiểm soát DL-Lite, đây là những ngôn ngữ bản thể học dễ kiểm soát phổ biến nhất trong Web ngữ nghĩa và cơ sở dữ liệu. Chúng tôi cũng sẽ trình bày cách thêm phủ định phân tầng vào Datalog ± trong khi vẫn giữ cho truy vấn bản thể học dễ kiểm soát. Hơn nữa, khả năng của họ Datalog ± thực sự đáng quan tâm và được dùng trong các khuôn khổ khác nhau như tích hợp dữ liệu và trao đổi dữ liệu. Nó tạo điều kiện để áp dụng các kết quả từ cơ sở dữ liệu cho lĩnh vực Web ngữ nghĩa.

Nội dung

1 Introduction	1. Giới thiệu
Ontology languages, rule-based systems, and their integrations play a central role in the development of the Semantic Web. Although there are a plethora of approaches to tight and loose (or hybrid) integrations of ontology languages and rule-based systems, and to generalizations of ontology languages by the ability to express rules, there is literally no previous work on how to generalize database rules and dependencies so that they can express ontological axioms. This is surprising, especially also because there are recently strong interests in the Semantic Web community on highly scalable formalisms for the Web of Data, which would benefit very much from applying technologies and results from databases.	[redacted]
In this paper, we try to fill this gap. We propose and study variants of	[redacted]

Datalog that are suited for efficient ontological reasoning, and, in particular, for tractable ontology-based query answering. We introduce the Datalog± family of Datalog variants, which extend plain Datalog by the possibility of existential quantification in rule heads, and by a number of other features, and, at the same time, restrict the rule syntax in order to achieve tractability. The goal of this paper is threefold:

• First, we aim at bridging an apparent gap in expressive power between database query languages and description logics (DLs) as ontology languages, extending the well-known Datalog language in order to embed DLs.

• Second, we aim at transferring important concepts and proof techniques from database theory to DLs. For example, it was so far not clear how to enrich tractable DLs by the feature of nonmonotonic negation. By the results of the present paper, we are now able to enrich DLs by stratified negation via mappings from DLs to Datalog± with stratified negation.

• Last but not least, we have a genuine interest in studying new fascinating tractable query languages. We are convinced that these languages are of independent relevance and interest, even without refer¬ence to ontological reasoning. Moreover, we have reasons to believe that the languages that we discuss may be useful for data exchange [49], and constraint satisfaction for automatic configuration, where value invention

[redacted]

[redacted]

[redacted]

[redacted]

techniques are used [50, 80]. For lack of space, we do not discuss these applications in detail here.	[redacted]
In addition to playing a key role in the development of the Semantic Web, ontologies are also becoming more and more important in the database area, for example, in data modeling and information integra¬tion [71]. While much of the research on DLs of the last decade was centered around decidability issues, there is a current trend towards highly scalable procedures for query answering over ontologies. A family of well-known DLs fulfilling these criteria is, e.g., the DL-Lite family [34, 89] (which has recently been further extended in [8, 9]). The following example briefly illustrates how queries can be posed and answered in DL-Lite.	[redacted]
Example 1 A DL knowledge base consists of a TBox and an ABox. For example, the knowledge that every conference paper is an article and that every scientist is the author of at least one paper can be expressed by the axioms ConferencePaper C Article and Scientist C BisAuthorOf in the TBox, respectively, while the knowledge that John is a scientist can be expressed by the axiom Scientist(john) in the ABox. A simple Boolean conjunctive query (BCQ) asking whether John authors a paper is BX isAuthorOf(john, X).	[redacted]
An ABox can be identified with an extensional database, while a TBox can be regarded as a set of integrity constraints involving, among others, functional dependencies and (possibly recursive) inclusion	[redacted]

dependencies [48, 1]. An important result of [34, 89] is that the DL-Lite description logics, in particular, DL-Lite^, DL-LiteR, and DL-Litea, are not only decidable, but that answering (unions of) conjunctive queries for them is in LOGSPACE, and actually in ACo, in the data complexity, and query answering in DL-Lite is FO-rewritable (see below) [34].	[redacted]
In the context of DLs, data complexity is the complexity of query answering over input ABoxes, when both the TBox and the query are fixed. This scenario is very similar to query answering with well-known rule-based languages, such as Datalog. It is easy to see that plain Datalog can neither directly express DL- Lite disjointness constraints (e.g., ConferencePaper C -JournalPaper), nor the functional constraints used in DL-LiteF (e.g., (funct hasFirstAuthor)). Moreover, as observed in [87], the lack of value creation makes plain Datalog not very well suited for ontological reasoning with inclusion axioms either (e.g., Scientist C BisAuthorOf). It is thus natural to ask whether Datalog can be suitably modified to nicely accommodate ontological axioms and constraints such as those expressible in the DL-Lite family. In particular, we have addressed the following two questions:	[redacted] [redacted]
Question 1: What are the main modifications of Datalog that are required for ontological knowledge rep-resentation and query-answering?	[redacted]
Question 2: Are there versions of	[redacted]

Datalog that encompass the DL-Lite family of description logics, and that share the favorable data complexity bounds for query-answering with DL-Lite? If so, how do they look like?	[redacted]
As an answer to Question 1, we identified the possibility of having existentially quantified variables in rule heads as the main Datalog extension enabling ontological knowledge representation and reasoning. Datalog rules extended this way are known as tuple generating dependencies (TGDs), see [17]. Given that fact inference (let alone conjunctive query answering) under TGDs is undecidable [16, 2], we must somehow restrict the rule syntax for achieving decidability. We thus require that the rule bodies of TGDs are guarded. This means that in each rule body of a TGD there must exist an atom, called guard, in which all non-existentially quantified variables of the rule occur as arguments. An example of a guarded TGD is P(X) A R(X, Y) A Q(Y) ^ BZ R(Y, Z).	[redacted]
Guarded TGDs form the first Datalog± formalism that we consider. Note that this formalization was briefly mentioned in [23]. We embark in Section 3 in a detailed analysis of the data complexity of this formalism. To this aim, we study the behavior of the (oblivious) chase algorithm [79, 17], a well-known al¬gorithm for constructing a (usually infinite) universal model chase (D, £) of a given extensional database D and a set of guarded TGDs £.	[redacted]
As a key lemma, we prove that for	[redacted]

each set of guarded TGDs £, there exists a constant 7 such that for every extensional database D, whenever a ground atom a is generated while chasing D with £, then the ground atom a and a whole derivation of a from D and £ must be generated at depth at most 7. Using this lemma, we can show that whenever a Boolean conjunctive query (BCQ) Q maps homomorphically into chase(D, £), then it maps into the initial fragment of constant depth k x |Q| of chase(D, £). This result is a nontrivial generalization of a classical result by Johnson and Klug [58] on inclusion dependencies, which are a restricted class of guarded TGDs. For the complexity of fact inference and answering BCQs, we then get the following result:

[redacted]

Theorem: Given a database D and a fixed set of guarded TGDs £, deciding whether D U £ |= a for facts a is PTIME-complete and can be done in linear time. Moreover, deciding whether D U £ = Q is not harder than BCQ evaluation over extensional databases (without guarded TGDs).

[redacted]

Guarded TGDs are sufficiently expressive to model the tractable description logic EL [10, 11] (as well as the more expressive ELIf [62]; see Section 9.2), but are still more expressive than actually necessary for modeling DL-Lite. Therefore, in Section 4, we consider the further restricted class of linear TGDs. These consist of TGDs whose bodies contain only single atoms (and so are trivially guarded, or TGDs whose bodies contain only guards, called

[redacted]

multi-linear TGDs). Note that this class generalizes the class of inclusion dependencies. A detailed analysis of chase properties of linear TGDs yields the following results.

Theorem: Given a database D, a fixed set of linear TGDs £, and a fixed BCQ Q, deciding whether D U £ |= Q is in ACo. In particular, this problem is FO-rewritable, i.e., Q and £ can be compiled into a first-order formula 0 such that for each database D, it holds that D U £ = Q iff D = 0.

In order to capture DL-Lite, we further enrich guarded Datalog by two additional other features: negative constraints and keys. A negative constraint is a Horn clause whose body is not necessarily guarded and whose head is the truth constant false which we denote by ±. For example, the requirement that a person ID cannot simultaneously appear in the employee (ID, Name) and in the retired (ID, Name) relation can be expressed by:

employee (X, Y) A retired(X, Z) ^ ± .

While negative constraints do add expressive power to Datalog, they are actually very easy to handle, and we show that the addition of negative constraints does not increase the complexity of query answering. We also allow a limited form of equality-generating dependencies, namely, keys, to be specified, but we require that these keys be - in a precise sense - not conflicting with the existential rules of the Datalog program. We lift a result from [30] about non-key-conflicting inclusion dependencies to

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

the setting of arbitrary TGDs to prove that the keys that we consider do not increase the complexity. With these additions we have a quite expressive and still extremely efficient version of Datalog±.	[redacted]
Theorem: Query answering with Datalog± based on guarded TGDs (resp., linear TGDS), negative constraints, and keys that do not conflict with the TGDs is possible in polynomial time in the data complexity (resp., FO-rewritable).	[redacted]
Let us refer to the above basic version of Datalog± (linear TGDs, negative constraints, and non-conflic¬ting keys) as Datalog±, and to the guarded version with negative constraints and non-conflicting keys as Datalog±. We are finally able to show in Sections 7 to 9 that all description logics of the well-known DL- Lite family of description logics [34] smoothly translate into Datalog±. The relationships between Datalog±, Datalog±, DL-Lite, and EL are summarized in Fig. 1.	[redacted]
Theorem: The description logics DL-LiteX of the DL-Lite family and their extensions with n-ary relations DLR-LiteX can all be reduced to Datalog±.	[redacted]
Example 2 The axioms of the TBox of Example 1 are translated to the TGDs ConfPaper(X) ^ Article(X) and Scientist(X) ^ BZ isAuthorOf(X, Z), while the axiom of the ABox is translated to the database atom Scientist(john).	[redacted]
The translation from the DL-Lite family into Datalog± is so smooth and natural, that Datalog± can rightly be called a DL. Note that Datalog± is	[redacted]

strictly more expressive than any of the description logics of the DL-Lite family. Interestingly, we prove that (at most binary) linear TGDs alone can express useful

Figure 1: Relationships between Datalog±, Datalog±, DL-Lite, and EL.

ontological relationships such as, e.g., manager(X) ^ manages(X, X) that are not expressible in any of the description logics of the DL-Lite family.

In the DL community, there is currently a need for enhancing tractable DLs by some nonmonotonic negation (where negative information is derived from the absence of positive one). It was asked whether there is some stratified negation for DLs. Given our translation from DL-Lite to Datalog±, this amounts to ask whether there is a satisfactory stratified negation for Datalog±, and, in particular:

Question 3: Can we extend the concept of safe stratified negation to guarded TGDs?

In classical Datalog with stratified negation [7], each stratum is finite, and the stratum i + 1 can be evaluated as soon as all facts in stratum i have been derived. With guarded TGDs, this is not so. Given that usually an infinite number of facts is generated by the chase, each stratum, including the lowest may be infinite, which means that single strata may at no time be fully computed. The difficulty is then, how long to wait before deciding that

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

a negative atom in a rule body is satisfied. We solve this problem by making use of the above-mentioned constant-depth bounds for atom derivations. We define a new version of the chase that uses a constant-depth bound for establishing whether a negative atom whose arguments all appear in those of a (positive) rule guard is satisfied. We show that this semantics is stratification-independent, corresponds to a perfect model semantics, and that query answering can be done in polynomial time for guarded TGDs and is FO-rewritable for linear TGDs.

The rest of this paper is organized as follows. In Section 2, we give some preliminaries and basic defi¬nitions. Sections 3 and 4 deal with guarded Datalog± and the special case of linear Datalog±, respectively. In Section 5, we show how negative constraints can be added. In Section 6, we discuss the addition of keys. Sections 7 to 9 deal with the translation of the DL-Lite family to Datalog±, while Section 10 defines stratified Datalog±. In Section 11, we discuss related work. Section 12 summarizes the main results and gives an outlook on future research. Note that detailed proofs of all results are given in Appendices A to G.

2 Preliminaries

In this section, we briefly recall some basics on relational databases, conjunctive queries (CQs), Boolean conjunctive queries (BCQs), tuple-generating dependencies (TGDs), and the chase procedure relative to such dependencies.

2.1 Databases and Queries

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

As for the elementary ingredients, we assume constants, nulls, and variables as follows; they serve as ar¬guments in atomic formulas in databases, queries, and dependencies. We assume (i) an infinite universe of (data) constants A (which constitute the “normal” domain of a database), (ii) an infinite set of (labeled) nulls AN (used as “fresh” Skolem terms, which are placeholders for unknown values, and can thus be seen as variables), and (iii) an infinite set of variables X (used in queries and dependencies). Different con¬stants represent different values (unique name assumption), while different nulls may represent the same value. We assume a lexicographic order on A U AN, with every symbol in AN following all symbols in A. We denote by X sequences of variables Xi,..., Xk with k ^ 0.

We next define atomic formulas, which occur in databases, queries, and dependencies, and which are constructed from relation names and terms, as usual. We assume a relational schema R, which is a finite set of relation names (or predicate symbols, or simply predicates) along with the names of the attributes of each relation. A position P[i] identifies the i-th argument of a predicate P. A term t is a constant, null, or variable. An atomic formula (or atom) a has the form P (t1, ...,tn), where P is an n-ary predicate, and t1,tn are terms. We denote by pred(a) and dom(a) its predicate and the set of all its arguments, respectively. The latter two notations are naturally extended to sets of atoms and

[redacted]

[redacted]

conjunctions of atoms. A conjunction of atoms is often identified with the set of all its atoms.

We are now ready to define the notion of a database relative to a relational schema, as well as the syntax and the semantics of conjunctive and Boolean conjunctive queries to databases. A database (instance) D for a relational schema R is a (possibly infinite) set of atoms with predicates from R and arguments from A. A conjunctive query (CQ) over R has the form Q(X) = BY $(X, Y), where $(X, Y) is a conjunction of atoms with the variables X and Y, and eventually constants, but without nulls. Note that $(X, Y) may also contain equalities but no inequalities. A Boolean CQ (BCQ) over R is a CQ of the form Q(). We often write a BCQ as the set of all its atoms, having constants and variables as arguments, and omitting the quantifiers. Answers to CQs and BCQs are defined via homomorphisms, which are mappings ^: A U AN U X —— A U AN U X such that (i) c € A implies ^(c) = c, (ii) c € AN implies ^(c) € A U AN, and (iii) ^ is naturally extended to atoms, sets of atoms, and conjunctions of atoms. The set of all answers to a CQ Q(X) = BY $(X, Y) over a database D, denoted Q(D), is the set of all tuples t over A for which there exists a homomorphism ^: X U Y — A U AN such that ^($(X, Y)) C D and ^(X) = t. The answer to a BCQ Q() = BY $(Y) over a database D is Fes, denoted D |= Q, iff Q(D) = 0, i.e., there exists a homomorphism ^: Y — A U AN

[redacted]

[redacted]

[redacted]

such that ^($(Y)) C D.

Example 3 Consider an employee database, which stores information about managers, employees, and departments, where managers may supervise employees and direct departments, and employees may work in a department. The relational schema R consists of the unary predicates manager and employee as well as the binary predicates directs, supervises, and works-in with obvious semantics. A database D for R is given as follows:

D = {employee (jo), manager (jo), directs (jo, finance), supervises (jo, ada), employee (ada), worksJn (ada, finance)} .

It encodes that Jo is an employee and a manager directing the finance department and supervising Ada, who is an employee working in the finance department. A CQ is given by Q(X) = manager(X) A directs (X, finance), which asks for all managers directing the finance department, while a BCQ is given

by Q() = BX (manager(X) A directs(X, finance)), often simply abbreviated as the set of atoms {mana¬ger (X), directs(X, finance)}, which asks whether there exists a manager directing the finance department. The set of all answers to the former over D is given by Q(D) = {jo}, while the answer to the latter is Yes.

2.2 Tuple-Generating Dependencies (TGDs)

Tuple-generating dependencies (TGDs) describe constraints on databases in the form of generalized Datalog rules with existentially quantified conjunctions of atoms in

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

rule heads; their syntax and semantics are as follows. Given a relational schema R, a tuple-generating dependency (TGD) a is a first-order formula of the form VXVY $(X, Y) ^ BZ ^(X, Z), where $(X, Y) and ^(X, Z) are conjunctions of atoms over R (without nulls), called the body and the head of a, denoted body (a) and head (a), respectively. We usually omit the universal quantifiers in TGDs. Such a is satisfied in a database D for R iff, whenever there exists a homomorphism h that maps the atoms of $(X, Y) to atoms of D, there exists an extension h' of h that maps the atoms of ^(X, Z) to atoms of D. All sets of TGDs are finite here.

[redacted]

Example 4 Consider again the employee database D of Example 3. Some constraints on D along with their encoding as TGDs (where we use “,” to denote the Boolean conjunction “A”) are as follows:

[redacted]

• every manager is an employee: manager(M) ^ employee (M);

[redacted]

• every manager directs at least one department:
manager(M) ^ BP directs(M, P);

[redacted]

• every employee who directs a department is a manager, and supervises at least another employee who works in the same department:
employee(E), directs(E, P) ^ BE' manager(E), supervises(E, E'), works-in(E', P);

[redacted]

• every employee supervising a manager is a manager:
employee(E), supervises(E, E'), manager(E') ^ manager(E).

[redacted]

It is not difficult to verify that all the

[redacted]

above TGDs are satisfied in D. Consider next the database D' defined as follows: D' = D U {manager(ada)} . Then, the first, the third, and the last TGD listed above are all satisfied in D', while the second TGD is not satisfied in D'.	[redacted]
Query answering under TGDs, i.e., the evaluation of CQs and BCQs on databases under a set of TGDs is defined as follows. For a database D for R, and a set of TGDs £ on R, the set of models of D and £, denoted mods(D, £), is the set of all (possibly infinite) databases B such that (i) D C B and (ii) every a € £ is satisfied in B. The set of answers for a CQ Q to D and £, denoted ans(Q, D, £), is the set of all tuples a such that a € Q(B) for all B € mods(D, £). The answer for a BCQ Q to D and £ is Yes, denoted D U £ = Q, iff ans(Q, D, £) = 0, i.e., B = Q for every B € mods(D, £). Note that query answering under general TGDs is undecidable [16], even when the schema and TGDs are fixed [23].	[redacted]
Example 5 Consider again the employee databases D and D' of Examples 3 and 4, respectively, and the set of TGDs £ of Example 4. Then, D is a model of D and £, i.e., D € mods(D, £), while D' is not a model of D' and £, i.e., D' € mods(D', £), since the second and the third TGD of Example 4 are not satisfied in D'. Trivially, every model of D' and £ is a superset of D'. In particular, the following databases Bi, B2, and B3 are models of D' and £: B1 = D' U {directs(ada, finance), supervises(ada, ada)} ,	[redacted]

B2 = D' U {directs (ada, finance), supervises (ada, bill), worksJn(bill, finance)} ,

B3 = D' U {directs (ada, toy), supervises (ada, bill), worksJn (bill, toy)} .

On the contrary, the following database B4 is not a model of D' and £, since the third TGD of Example 4 is not satisfied in B4:

B4 = D' U {directs (ada, toy), supervises (ada, tom)} .

Notice that the atom employee (jo) is true in all models of D' and £; therefore, the BCQ {employee (jo)} evaluates to true over D' and £. This also holds for the BCQ {directs(ada,X)}, while the BCQ {directs(ada, finance)} evaluates to false over D' and £, since it is false in the database B3.

We recall that the two problems of CQ and BCQ evaluation under TGDs are LOGSPACE-equivalent [35, 58,49, 41]. Moreover, it is easy to see that the query output tuple (QOT) problem (as a decision version of CQ evaluation) and BCQ evaluation are AC0-reducible to each other. Henceforth, we thus focus only on the BCQ evaluation problem. All complexity results carry over to the other problems. We also recall that query answering under TGDs is equivalent to query answering under TGDs with only singleton atoms in their heads. In the sequel, we thus always assume w.l.o.g. that every TGD has a singleton atom in its head.

2.3 The TGD Chase

The chase was introduced to enable checking implication of dependencies [79], and later also for

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

checking query containment [58]. It is a procedure for repairing a database relative to a set of dependencies, so that the result of the chase satisfies the dependencies. By "chase", we refer both to the chase procedure and to its output. The TGD chase works on a database through so-called TGD chase rules (for an extended chase with also equality-generating dependencies (EGDs), see Section 6). The TGD chase rule comes in two flavors: restricted and oblivious, where the restricted one applies TGDs only when they are not satisfied (to repair them), while the oblivious one always applies TGDs (if they produce a new result). We focus on the oblivious one, since it makes proofs technically simpler. The (oblivious) TGD chase rule defined below is the building block of the chase.

TGD Chase Rule. Consider a database D for a relational schema R, and a TGD a on R of the form $(X, Y) ^ BZ ^(X, Z). Then, a is applicable to D if there exists a homomorphism h that maps the atoms of $(X, Y) to atoms of D. Let a be applicable to D, and h1 be a homomorphism that extends h as follows: for each Xi € X, h1(Xi) = h(Xi); for each Zj € Z, h1(Zj) = Zj, where Zj is a "fresh" null, i.e., Zj € An, Zj does not occur in D, and Zj lexicographically follows all other nulls already introduced. The application of a on D adds to D the atom h1(^(X, Z)) if not already in D (which is possible when Z is empty). ■

The chase algorithm for a database D

[redacted]

[redacted]

[redacted]

[redacted]

and a set of TGDs £ consists of an exhaustive application of the TGD chase rule in a breadth-first (level-saturating) fashion, which leads as result to a (possibly infinite) chase for D and £. Formally, the chase of level up to 0 of D relative to £, denoted chase 0(D, £), is defined as D, assigning to every atom in D the (derivation) level 0. For every k ^ 1, the chase of level up to k of D relative to £, denoted chasek(D, £), is constructed as follows: let I1,..., In be all possible images of bodies of TGDs in £ relative to some homomorphism such that (i) I1,..., In C chasek-i(D, £) and (ii) the highest level of an atom in every Ii is k — 1; then, perform every corresponding TGD application on chasek-1 (D, £), choosing the applied TGDs and homomorphisms following a deterministic execution strategy (e.g., using a (fixed) linear and lexicographic order for the TGDs and homomorphisms, respectively), and assigning to every new atom the (derivation) level k. The chase of D relative to £, denoted chase(D, £), is then defined as the limit of chasek(D, £) for k — to.

The (possibly infinite) chase relative to TGDs is a universal model, i.e., there exists a homomorphism from chase(D, £) onto every B € mods(D, £) [41, 23]. This result implies that BCQs Q over D and £ can be evaluated on the chase for D and £, i.e., D U £ |= Q is equivalent to chase(D, £) |= Q.

Example 6 Consider again the employee database D' of Example 3 and the set of TGDs £ of Example 4. Then, in the construction of chase

[redacted]

[redacted]

[redacted]

(D', £), we apply first the second TGD of Example 4 to manager (jo) (resp., manager (ada)), adding directs (jo, z1) (resp., directs (ada, z2)), where z1 and z2 are “fresh” nulls, and then the third TGD to employee (jo) and directs (jo, z1) (resp., employee (ada) and directs (ada, z2)), adding supervises (jo , z3) and worksJn (z3,z1) (resp., supervises (ada , z4) and worksJn (z4,z2)), where z3 and z4 are “fresh” nulls. Hence, the construction yields a finite chase chase (D', £), given as follows:	[redacted]
chase (D', £) = D' U {directs (jo, z1), directs (ada, z2), supervises (jo, z3), worksJn (z3, z1), supervises (ada, z4), worksJn (z4, z2)} .	[redacted]
Here, every atom in D' is of level 0, the two atoms directs (jo, z1) and directs (ada, z2) are of level 1, and the other four atoms are of level 2.	[redacted]
3 Guarded Datalog±	[redacted]
We now introduce guarded Datalog± as a class of special TGDs that exhibit computational tractability in the data, while being at the same time expressive enough to model ontologies. BCQs relative to such TGDs can be evaluated on a finite part of the chase, which is of constant size when the query and the TGDs are fixed. Based on this result, the data complexity of evaluating BCQs relative to guarded TGDs turns out to be polynomial in general and linear for atomic queries.	[redacted]
A TGD a is guarded iff it contains an atom in its body that contains all universally quantified variables of a. The leftmost such atom is the guard atom (or guard) of a. The non-guard	[redacted]

atoms in the body of a are the side atoms of a. For ease of presentation, we assume that guarded TGDs contain no constants (but all the results of this paper can be extended to guarded TGDs without this restriction).

Example 7 The TGD r(X, Y), s(Y, X, Z) — BW s(Z, X, W) is guarded (where s(Y, X, Z) is the guard, and r(X, Y) is a side atom), while the TGD r(X, Y), r(Y, Z) — r(X, Z) is not guarded, since it has no guard, i.e., no body atom contains all the (universally quantified) variables in the body. Furthermore, it is easy to verify that every TGD in Example 4 is guarded.

Figure 2: Chase graph (left side) and guarded chase forest (right side) for Example 8.

Note that sets of guarded TGDs (with single-atom heads) are theories in the guarded fragment of first-order logic [4]. Note also that guardedness is a truly fundamental class ensuring decidability. As shown in [23] , adding a single unguarded Datalog rule to a guarded Datalog± program may destroy decidability.

In the sequel, let R be a relational schema, D be a database for R, and £ be a set of guarded TGDs on R. We first give some preliminary definitions as follows. The chase graph for D and £ is the directed graph consisting of chase(D, £) as the set of nodes and having an arrow from a to b iff b is obtained from a and possibly other atoms by a one-step application of a TGD a € £.

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Here, we mark a as guard iff a is the guard of a. The guarded chase forest for D and £ contains (i) for every atom a € D, one node labeled with a, and (ii) for every node labeled with a € chase(D, £) and for every atom b € chase(D, £) that is obtained from a and possibly other atoms by a one-step application of a TGD a € £ with a as guard, one node labeled with b along with an arrow from the node labeled with a. The subtree of a node v labeled with an atom a (also simply called subtree of a) in this forest, denoted subtree (a), is the restriction of the forest to all descendants of v. The type of an atom a, denoted type (a), is the set of all atoms b in chase (D, £) that have only constants and nulls from a as arguments. Note that the subtree of (a node labeled with) an atom a in the guarded chase forest depends only on the set of all atoms in the type of a (and no others).

[redacted]

Example 8 Consider the database D = {r(a, b),s(b)} and the set of TGDs £ consisting of the following two TGDs ai and a2:

[redacted]

ai: r(X,Y),s(Y) ^ 3Zr(Z,X), a2 : r(X,Y) ^ s(X).

[redacted]

The first part of the (infinite) chase graph (resp., guarded chase forest) for D and £ is shown in Fig. 2, left (resp., right) side, where the arrows have the applied TGDs as labels (formally not a part of the graph (resp., forest)). The number on the upper right side of every atom indicates the derivation level of the atom. The subtree of (the node labeled with) r(z1, a) in the guarded chase forest is also shown in Fig. 2, right side. The type of r(z1,a) consists of the atoms r(z1,a), s(a),

[redacted]

and s(z1).	
Given a finite set S C A U AN, two sets of atoms A1 and A2 are S-isomorphic (or isomorphic if S = 0) iff a bijection fi: A1 U dom(A1) ^ A2 U dom(A2) exists such that (i) fi and fi-1 are homomorphisms, and (ii) fi(c) = c = fi-1(c) for all c € S. Note that fi is already fully determined by its restriction to dom(A1). Two atoms a1 and a2 are S-isomorphic (or isomorphic if S = 0) iff {a1} and {a2} are S-isomorphic.	[redacted]
The notion of S-isomorphism (or isomorphism if S = 0) is naturally extended to more complex structures, such as pairs of two subtrees (V1, E1) and (V2, E2) of the guarded chase forest, and two pairs (b1, S1) and (b2, S2), where b1 and b2 are atoms, and S1 and S2 are sets of atoms.	[redacted]
Example 9 The two sets of atoms {a1: r(a, z1, z2), a2 : s(b, z2), a3 : t(z3, z4} and {b1: r(a, z1, z5), b2 : s(b, z5), b3 : t(z6, z7)}, where a, b € A and z1,..., z7 € AN, are {a, z1}-isomorphic via the bijection P de¬fined by P(ai) = bi, i € {1, 2, 3}, P(z2) = z5, P(z3) = z6, P(z4) = z7, and P(c) = c for all other c € A U AN. Furthermore, let a = r(a, b, z1), where a, b € A and z1 € AN. Then, s(b, z3) and s(b, z4) are dom(a)-isomorphic, while s(b, z3) and s(b, z1) are not (with z3, z4 € AN).	[redacted]
The following lemma shows that if two atoms in the guarded chase forest for D and £ along with their types are S-isomorphic, then their subtrees are also S-isomorphic, which can be proved by induction on the number of applications of the TGD chase rule to generate the	[redacted]

subtrees of the two atoms.

Lemma 1 Let R be a relational schema, D be a database for R, and £ be a set of guarded TGDs on R. Let S be a finite subset of A U AN, and let a1 and a2 be atoms from chase (D, £) such that (a1, type (a1)) and (a2, type(a2)) are S-isomorphic. Then, the subtree of a1 is S-isomorphic to the subtree of a2.

The next lemma provides, given an atom a € chase(D, £), an upper bound for the number of all non- dom(a)-isomorphic pairs consisting of an atom and a type with arguments from a and new nulls. The result follows from a simple combinatorial analysis of the number of all possible such pairs.

Lemma 2 Let R be a relational schema, D be a database for R, and £ be a set of guarded TGDs on R. Let w be the maximal arity of a predicate in R, 5 = | R | ■ (2w)w ■ 2|R|^2w)w, and a € chase (D, £). Let P be a set of pairs (b, S), each consisting of an atom b and its type S of atoms c with arguments from a and new nulls. If |P | > 5, then P contains at least two dom (a)-isomorphic pairs.

We next define the guarded depth of atoms in the guarded chase forest as follows. The guarded depth of an atom a in the guarded chase forest for D and £, denoted depth (a), is the smallest length of a path from (a node labeled with) some d € D to (a node labeled with) a in the forest. Note that this is in general different from the derivation level in the chase (see Example 10). The guarded chase of level up to k ^ 0 for D and £, denoted g-chasek(D, £), is the set of all atoms in the guarded chase forest

[redacted]

[redacted]

[redacted]

[redacted]

of depth at most k.

Example 10 Consider the database D = {r1(a, b)} and the set of TGDs £ consisting of the following three TGDs a1, a2, and a3:

a1: r3(X, Y) — r2(X) ,

a2 : r1(X,Y) — BZ r3(Y, Z),

a3 : n(X,Y),r2(Y) — n(Y,X).

The chase graph for D and £ is shown in Fig. 3. It nearly coincides with the guarded chase forest for D and £, where only the dashed arrow is removed. Every atom is also labeled with its guarded depth and its derivation level.

The next lemma shows that BCQs in the form of single ground atoms a can be evaluated using only a finite, initial portion of the guarded chase forest, whose size depends only on the relational schema R. In fact, the lemma even shows the stronger result that also a whole proof of a is contained in such a portion

Figure 3: Guarded depth / derivation level of atoms in the chase graph.

of the forest. Here, a proof of an atom a from D and £ is a subgraph n of the chase graph such that n contains the atom a as a vertex, and whenever n contains b as a vertex, where b € D, then n also contains

a set of “parent” atoms b1,..., br for b such that there exists a TGD in £ that applied to b1

b. The proof of Lemma 3 in Appendix A is done by induction on the derivation level of a. There, it is also stated how 7 is bounded in terms of the size of the relational schema R, namely double-exponentially in the general case, and single-exponentially in case of a fixed arity. Of course, for a fixed relational

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

schema R, 7 is a constant.

Lemma 3 Let R be a relational schema, D be a database for R, and £ be a set of guarded TGDs on R. Then, there is a constant 7, depending only on R, such that for each ground atom a € chase (D, £), there is a proof of a from D and £, whose atoms all belong to g-chase1 (D, £).

[redacted]

The following lemma shows that BCQs Q can be evaluated using only a finite, initial portion of the guarded chase forest, whose size depends only on the query Q and the relational schema R. The result is proved by showing that every path from D to (the image of) a query atom in the guarded chase forest, whose length exceeds a certain value (depending on Q and R), has two atoms with dom (a)-isomorphic subtrees (since two atoms and their types are dom (a)-isomorphic), and thus Q can also be evaluated "closer" to D.

[redacted]

Lemma 4 Let R be a relational schema, D be a database for R, £ be a set of guarded TGDs on R, and Q be a BCQ over R. If there exists a homomorphism ^ that maps Q into chase (D, £), then there exists a homomorphism X that maps Q into g-chasek(D, £), where k depends only on Q and R.

[redacted]

Intuitively, the chase of a database relative to a set of guarded TGDs has a "periodicity" of atoms and their types, as illustrated by the following example.

[redacted]

Example 11 Every derivation level k ^ 2 of the chase graph for Example 8 in Fig. 2 is given by two atoms r(zk, zk-1) and s(zk-1), where the type of

[redacted]

the former is given by the three atoms r(zk, zk-1), s(zk-1), and s(zk). This “pattern” repeats indefinitely in the chase, as easily seen. For example, a BCQ Q = {r(X, Y), r(Z,X), r(W, Z)} will necessarily map onto three atoms that form a path in the guarded chase forest: however deep these atoms are in the chase, Q can anyway also be mapped onto the first levels, e.g., onto {r(z2, z1), r(z3, z2), r(z4, z3)}.	[redacted]
The above lemma informally says that whenever (homomorphic images of) the query atoms are con¬tained in the chase, then they are also contained in a finite, initial portion of the guarded chase forest, whose size is determined only by the query and the schema. However, it does not yet ensure that also a whole proof of the query atoms is contained in such a portion of the forest. This slightly stronger property is captured by the following definition.	[redacted]
Definition 1 Let R be a relational schema, and £ be a set of TGDs on R. Then, £ has the bounded guard-depth property (BGDP) iff, for every database D for R and for every BCQ Q, whenever there exists a homomorphism ^ that maps Q into chase(D, £), then there exists a homomorphism A of this kind such that a proof of every a € A(Q) from D and £ is contained in g-chaselg (D, £), where Yg depends only on Q and R.	[redacted]
The next theorem shows that, in fact, guarded TGDs have also this stronger bounded guard-depth prop¬erty. The proof of this result is based on the above Lemmas 3 and 4, where the former now also assures	[redacted]

that all side atoms that are necessary in a proof of the query atoms are contained in a finite, initial portion of the guarded chase forest, whose size is determined only by Q and R (which is slightly larger than the one for the query atoms only).

Theorem 5 Guarded TGDs enjoy the BGDP.

By this theorem, deciding BCQs in the guarded case is in P in the data complexity (where all but the database is fixed) [23]. It is also hard for P, as can be proved by reduction from propositional logic programming.

Theorem 6 Let R be a relational schema, D be a database for R, £ be a set of guarded TGDs on R, and Q be a BCQ over R. Then, deciding D U £ |= Q is P-complete in the data complexity.

Deciding Boolean atomic queries in the guarded case can even be done in linear time in the data complexity, as the following theorem shows, which holds by a reduction to propositional logic programming. Note that since general BCQs are not necessarily guarded, they are in general not reducible to atomic queries.

Theorem 7 Let R be a relational schema, D be a database for R, £ be a set of guarded TGDs on R, and Q be a Boolean atomic query over R. Then, deciding D U £ |= Q can be done in linear time in the data complexity.

4 Linear Datalog±

We now introduce linear Datalog± as a variant of guarded Datalog±, where query answering is even FO-rewritable in the data complexity. Nonetheless, (an extension of) linear

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Datalog± is still expressive enough for representing ontologies, as we will show in Sections 7 and 8 for ontologies encoded in the description logics of the DL-Lite family (DL-LiteF , DL-LiteR, and DL-Lite A [34, 89]).

[redacted]

A guarded TGD is linear iff it contains only a singleton body atom (i.e., the TGD is of the form VXVY $(X, Y) — BZ tf(X, Z), where $(X, Y) is an atom).

[redacted]

Example 12 Consider again the TGDs of Example 4. As easily verified, the first two are linear, while the last two are not. Another linear TGD is directs (E, P) — employee (E), restricting the first argument of the directs relation to employees.

[redacted]

Observe that linear Datalog± generalizes the well-known class of inclusion dependencies, and that this generalization is strict, for example, the linear TGD supervises(X, X) ^ manager(X), which asserts that all people supervising themselves are managers, is not expressible with inclusion dependencies.

[redacted]

We next define the bounded derivation-depth property for sets of TGDs, which is strictly stronger than the bounded guard-depth property (see Definition 1), since the former implies the latter, but not vice versa. Informally, the bounded derivation-depth property says that whenever (homomorphic images of) the query atoms are contained in the chase, then they (along with their derivations) are also contained in a finite, initial portion of the chase graph (rather than the guarded chase forest), whose size depends only on

[redacted]

the query and the schema. Definition 2 Let R be a relational schema, and £ be a set of TGDs on R. Then, we say that £ has the bounded derivation-depth property (BDDP) iff, for every database D for R and for every BCQ Q over R, whenever D U £ = Q, then chaseld (D, £) = Q, where Yd depends only on Q and R. Clearly, in the case of linear TGDs, for every a € chase (D, £), the subtree of a is now determined only by a itself, while in the case of guarded TGDs it depends on type (a). Therefore, for a single atom, its depth coincides with the number of applications of the TGD chase rule that are necessary to generate it. That is, the guarded chase forest coincides with the chase graph. Thus, by Theorem 5, we immediately obtain that linear TGDs have the bounded derivation-depth property. Corollary 8 Linear TGDs enjoy the BDDP. We next recall the notion of first-order rewritability for classes of TGDs. A class of TGDs C is first-order rewritable (or FO-rewritable) iff for every set of TGDs £ in C and for every BCQ Q, there exists a first-order query Qs such that, for every database D, it holds that D U £ = Q iff D = Qs. Since answering first-order queries is in ACo in the data complexity [99], also BCQ answering under FO-rewritable TGDs is in AC0 in the data complexity. The following result shows that BCQs Q relative to TGDs £ with the bounded derivation-depth property are FO-rewritable. The main ideas	[redacted] [redacted] [redacted] [redacted] [redacted]

behind its proof are informally summarized as follows. Since the deriva¬tion depth and the number of body atoms in TGDs in £ are bounded, the number of all database ancestors of query atoms is also bounded. So, the number of all non-isomorphic sets of potential database ancestors with variables as arguments is also bounded. Take the existentially quantified conjunction of every such ancestor set where Q is answered positively. Then, the FO-rewriting of Q is the disjunction of all these formulas.	[redacted]
Theorem 9 Let R be a relational schema, £ be a set of TGDs on R, and Q be a BCQ over R. If £ enjoys the BDDP, then Q is FO-rewritable.	[redacted]
As an immediate consequence of Corollary 8 and Theorem 9, we obtain that BCQs are FO-rewritable in the linear case.	[redacted]
Corollary 10 Let R be a relational schema, £ be a set of linear TGDs on R, and Q be a BCQ over R. Then, Q is FO-rewritable.	[redacted]
Observe that all the above results also apply to multi-linear TGDs, which are TGDs with only guards in their bodies, since here the guarded chase forest can be chosen in such a way that the depth of all its atoms coincides with their derivation depth. Formally, a TGD a is multi-linear iff all its body atoms have the same variables (i.e., a has the form VXVY $(X, Y) ^ 3Z ^(X, Z), where $(X, Y) is a conjunction of atoms Pi(X, Y), each containing each variable of X and Y).	[redacted]
5 Adding (Negative) Constraints In this section, we extend Datalog± by (negative) constraints, which are	[redacted]

an important ingredient, in particu¬lar, for representing ontologies.

[redacted]

A negative constraint (or simply constraint) is a first-order formula of the form VX $(X) — ±, where $(X) is a (not necessarily guarded) conjunction of atoms (without nulls). It is often also written as VX $'(X) — —p(X), where $'(X) is obtained from $(X) by removing the atom p(X). We usually omit the universal quantifiers, and we implicitly assume that all sets of constraints are finite here.

[redacted]

Example 13 If the two unary predicates c and c' represent two classes (also called concepts in DLs), we may use the constraint c(X), c'(X) — ± (or alternatively c(X) — — c'(X)) to assert that the two classes have no common instances. Similarly, if additionally the binary predicate r represents a relationship (also called a role in DLs), we may use c(X), r(X, Y) — ± to enforce that no member of the class c participates to the relationship r. Furthermore, if the two binary predicates r and r' represent two relationships, we may use the constraint r(X, Y), r'(X, Y) — ± to express that the two relationships are disjoint.

[redacted]

Query answering on a database D under a set of TGDs £T (as well as a set of EGDs £E as introduced in the next section) and a set of constraints £C can be done effortless by additionally checking that every constraint a = $(X) — ± € £C is satisfied in D and £T, each of which can be done by checking that the BCQ Qa = $(X) evaluates to false on D and £T. We write D U £T |= £C iff

[redacted]

every Qa with a € £C evaluates to false in D and £T. We thus obtain immediately the following result. Here, a BCQ Q is true in D and £T and £C, denoted D U £T U £C |= Q, iff (i) D U £T |= Q or (ii) D U £T |= £C (as usual in DLs).

[redacted]

Theorem 11 Let R be a relational schema, D be a database for R, £T and £C be sets of TGDs and constraints on R, respectively, and Q be a BCQ on R. Then, D U £T U £C |= Q iff(i) D U £T |= Q or(ii) D U £T |= Qa for some a € £C.

[redacted]

As an immediate consequence, we obtain that constraints do not increase the data complexity of answer¬ing BCQs in the guarded (resp., linear) case.

[redacted]

Corollary 12 Answering BCQs on databases under guarded (resp., linear) TGDs and constraints has the same data complexity as answering BCQs on databases under guarded (resp., linear) TGDs alone.

[redacted]

6 Adding Equality-Generating Dependencies (EGDs) and Keys

In this section, we add equality-generating dependencies (EGDs) to guarded (and linear) Datalog±, which are also important when representing ontologies. Note that EGDs generalize functional dependencies (FDs) and, in particular, key dependencies (or keys) [1]. In DL-Lite (see Sections 7 and 8), general EGDs cannot be formulated, but only keys. Therefore, we mainly focus on keys here. We transfer a result by [30] about non-key-conflicting (NKC) inclusion dependencies to the more general setting of guarded Datalog±.

[redacted]

However, while adding negative

[redacted]

constraints is effortless from a computational perspective, adding EGDs is more problematic: The interaction of TGDs and EGDs leads to undecidability of query answering even in simple cases, such that of functional and inclusion dependencies [36], or keys and inclusion dependencies (see, e.g., [30], where the proof of undecidability is done in the style of Vardi as in [58]). It can even be seen that a fixed set of EGDs and guarded TGDs can simulate a universal Turing machine, and thus query answering and even propositional ground atom inference is undecidable for such dependencies. For this reason, we consider a restricted class of EGDs, namely, non-conflicting key dependencies (or NC keys), which show a controlled interaction with TGDs (and negative constraints), such that they do not increase the complexity of answering BCQs. Nonetheless, this class is sufficient for modeling ontologies (e.g., in DL- Litej-, DL-LiteR, and DL-Lite A; see Lemmas 16 and 20).

[redacted]

An equality-generating dependency (or EGD) a is a first-order formula of the form VX $(X) ^ Xi = Xj, where $(X), called the body of a, denoted body (a), is a (not necessarily guarded) conjunc¬tion of atoms (without nulls), and Xi and Xj are variables from X. We call Xi = Xj the head of a, denoted head (a). Such a is satisfied in a database D for R iff, whenever there exists a homomorphism h such that h($(X)) C D, it holds that h(Xi) = h(Xj). We usually omit the universal quantifiers in EGDs, and all sets of EGDs are

[redacted]

finite here.	
Example 14 The following formula a is an equality-generating dependency: n(X,Y),r2(Y,Z) ^ Y = Z.	[redacted]
The database D = {r1(a, b),r2(b, b)} satisfies a, because every homomorphism h mapping the body of a to D is such that h(Y) = h(Z). On the contrary, the database D = {r1(a, b),r2(b, c)} does not satisfy a.	[redacted]
An EGD a on R of the form $(X) ^ Xj, = Xj is applicable to a database D for R iff there exists a homomorphism n: $(X) ^ D such that n(X%) and n(Xj) are different and not both constants. If n(X%) and n(Xj) are different constants in A, then there is a hard violation of a, and the chase fails. Otherwise, the result of the application of a to D is the database h(D) obtained from D by replacing every occurrence of a non-constant element e € {n(Xi), n(Xj)} in D by the other element e' (if e and e' are both nulls, then e precedes e' in the lexicographic order). Note that h is a homomorphism, but not necessarily an endomorphism of D, since h(D) is not necessarily a subset of D. But for the special class of TGDs and EGDs that we define in this section, h is actually an endomorphism of D.	[redacted]
The chase of a database D, in the presence of two sets £T and £E of TGDs and EGDs, respectively, denoted chase(D, £T U £E), is computed by iteratively applying (1) a single TGD once, according to the standard order and (2) the EGDs, as long as they are applicable (i.e., until a fixpoint is reached).	[redacted]
Example 15 Consider the following	[redacted]

set of TGDs and EGDs £ = {a1 ,a2, a3}: a1 : r(X,Y) ^ 3Zs(X,Y,Z), a2 : s(X, Y, Z) ^ Y = Z, a3 : r(X,Y),s(Z,Y,Y) ^ X = Y.	[redacted]
Let D be the database {r(a, b)}. In the computation of chase(D, £), we first apply a1 and add the fact s(a, b,z1), where z1 is a null. Then, the application of a2 on s(a, b, z1) yields z1 = b, thus turning s(a, b, z1) into s(a, b, b). Now, we apply a3 on r(a, b) and s(a, b, b), and by equating a = b, the chase fails; this is a hard violation, since both a and b are constants in A.	[redacted]
6.1 Separability The following definition generalizes the notion of separability originally introduced in [30] to Datalog±. Intuitively, the semantic notion of separability for EGDs formulates a controlled interaction of EGDs and TGDs / (negative) constraints, so that the EGDs do not increase the complexity of answering BCQs.	[redacted]
Definition 3 Let R be a relational schema, and £T and £E be sets of TGDs and EGDs on R, respectively. Then, £E is separable from £T iff for every database D for R, the following conditions (i) and (ii) are both satisfied:	[redacted]
(i) If there is ahard violation of an EGD of £E in chase (D, £T U £E), then there is also ahard violation of some EGD of £E in D.	[redacted]
(ii) If there is no chase failure, then for every BCQ Q, it holds that chase(D, £T U £E) |= Q iff chase(D, £T) |= Q.	[redacted]
Note that (ii) is equivalent to: (ii') if	[redacted]

there is no chase failure, then for every CQ Q, it holds that ans(Q, D, £T U £E) = ans(Q, D, £T). Here, (ii’) implies (ii), since (ii) is a special case of (ii’), and the converse holds, since a tuple t over A is an answer for a CQ Q to D and £ iff the BCQ Qt to D and £ evaluates to true, where Qt is obtained from Q by replacing each free variable by the corresponding constant in t. The following result shows that adding separable EGDs to TGDs and constraints does not increase the data complexity of answering BCQs in the guarded and linear case. It follows immediately from the fact that the separability of EGDs implies that chase failure can be directly evaluated on D. Here, for disjunctions of BCQs Q, D U £ |= Q iff D U £ |= Q for some BCQ Q in Q.

Theorem 13 Let R be a relational schema, £T and £E be fixed sets of TGDs and EGDs on R, respectively, where £E is separable from £T, and £C be a fixed set of constraints on R. Let QC be the disjunction of all Qa with a € £C. Then:

(a) If deciding D U £T |= Q V QC is feasible in polynomial time for each fixed query Q, then so is deciding D U £T U £E |= Q V QC.

(b) If deciding D U £T |= Q V QC is FO-rewritable for each fixed query Q, then so is deciding D U £T U £e |= Q V QC.

6.2 Non-Conflicting Keys

We next provide a sufficient syntactic condition for the separability of EGDs. We assume that the reader is familiar with the notions of a functional dependency (FD) (which informally encodes that

[redacted]

[redacted]

[redacted]

[redacted]

certain attributes of a relation functionally depend on others) and a key (dependency) (which is informally a tuple-identifying set of attributes of a relation) [1]. Clearly, FDs are special types of EGDs. A key K of a relation r can be written as a set of FDs that specify that K determines each other attribute of r. Thus, keys can be identified with sets of EGDs. It will be clear from the context when we regard a key as a set of attribute positions, and when we regard it as a set of EGDs. The following definition generalizes the notion of “non-key-conflicting” dependency relative to a set of keys, introduced in [30], to the context of arbitrary TGDS.	[redacted] [redacted]
Definition4 Let k beakey,and a be a TGD of the form \$(X, Y) — BZ r(X, Z). Then, k is non-conflicting (NC) with a iff either (i) the relational predicate on which k is defined is different from r, or (ii) the positions of k in r are not a proper subset of the X-positions in r in the head of a, and every variable in Z appears only once in the head of a. We say k is non-conflicting (NC) with a set of TGDs £T iff k is NC with every a € £t. A set of keys £K is non-conflicting (NC) with £T iff every k € £K is NC with £T.	[redacted]
Example 16 Consider the four keys k1, k2, k3, and k4 defined by the key attribute sets K1 = {r[1], r[2]}, K2 = {r[1],r[3]}, K3 = {r[3]}, and K4 = {r[1]}, respectively, and the TGD a = p(X, Y) — 3Z r(X, Y, Z). Then, the head predicate of a is r, and the set of positions in r with universally quanti¬fied variables is H = {r [1], r [2]}. Observe that all keys but k4 are	[redacted]

NC with a, since only K4 C H. Roughly, every atom added in a chase by applying a would have a fresh null in some position in K1, K2, and K3, thus never firing k1, k2, and k3, respectively.	[redacted] [redacted]
The following theorem shows that the property of being NC between keys and TGDs implies their separability. This generalizes a useful result of [30] on inclusion dependencies to the much larger class of all TGDs. The main idea behind the proof can be roughly described as follows. The NC condition between a key K and a TGD a assures that either (a) the application of a in the chase generates an atom with a fresh null in a position of k, and so the fact does not violate K (see also Example 16), or (b) the X-positions in the predicate r in the head of a coincide with the key positions of K in r, and thus any newly generated atom must have fresh distinct nulls in all but the key position, and may eventually be eliminated without violation. It then follows that the full chase does not fail. Since the new nulls are all distinct, it also contains a homomorphic image of the TGD chase. Therefore, the full chase is in fact homomorphically equivalent to the TGD chase.	[redacted] [redacted]
Theorem 14 Let R be a relational schema, £T and £K be sets of TGDs and keys on R, respectively, such that £K is NC with £T. Then, £K is separable from £T.	[redacted]
We conclude this section by stating that in the NC case, keys do not increase the data complexity of answering BCQs under guarded	[redacted]

(resp., linear) TGDs and constraints. This result follows immediately from Theorems 14 and 13.	[redacted]
Corollary 15 Let R be a relational schema, £T and £K be fixed sets of TGDs and keys on R, respectively, where £K is NC with £T, and £c be a fixed set of constraints on R. Let Qc be the disjunction of all Qa such that a € £c. Then:	[redacted]
(a) If £T are guarded TGDs, then deciding D U £T U £K |= Q V Qc is feasible in polynomial time. (b) If £T are linear TGDs, then deciding D U £T U £K |= Q V Qc is FO-rewritable.	[redacted]
7 Ontology Querying in DL-Lite F and DL-LiteR In this section, we show that the description logics DL-LiteF and DL-LiteR of the DL-Lite family [34] can both be reduced to linear (or multi-linear) Datalog± with (negative) constraints and NC keys, called Datalog±, and that the former are strictly less expressive than the latter. More specifically, we show how Datalog± can be used for answering BCQs in DL-LiteF and DL-LiteR ontologies. We first recall the syntax and the semantics of DL-LiteF and DL-LiteR. We then define the translation and provide the representation and expressivity results.	[redacted]
Note that DL-LiteR is able to fully capture the (DL fragment of) RDF Schema [19], the vocabulary description language for RDF; see [39] for a translation. Hence, Datalog± is also able to fully capture (the DL fragment of) RDF Schema.	[redacted]
The other description logics of the DL-Lite family [34] can be similarly	[redacted]

translated into Datalog±: the translation of DL-Lite A into Datalog± is given in Section 8, and the translations for the other DLs are sketched in Section 9. Note that it is mainly for didactic reasons that we start with the simpler DL-LiteF and DL-LiteR, and we continue with the slightly more complex DL-Lite A.	[redacted]
Intuitively, DLs model a domain of interest in terms of concepts and roles, which represent classes of individuals and binary relations on classes of individuals, respectively. A DL knowledge base (or ontology) encodes in particular subset relationships between concepts, subset relationships between roles, the membership of individuals to concepts, the membership of pairs of individuals to roles, and functional dependencies on roles.	[redacted]
7.1 Syntax of DL-LiteF and DL-LiteR	[redacted]
We now recall the syntax of DL-LiteF (also simply called DL-Lite). As for the elementary ingredients, we assume pairwise disjoint sets of atomic concepts, abstract roles, and individuals A, Ra, and I, respectively.	[redacted]
These elementary ingredients are used to construct roles and concepts, which are defined as follows: A basic role Q is either an atomic role P € Ra or its inverse P-. A (general) role R is either a basic role Q or the negation of a basic role —Q. A basic concept B is either an atomic concept A € A or an existential restriction on a basic role Q, denoted BQ. A (general) concept C is either a basic concept B or the negation of a basic concept — B.	[redacted]

Statements about roles and concepts are expressed via axioms, where an axiom is either (1) a con¬cept inclusion axiom B C C, where B is a basic concept, and C is a concept, or (2) a functionality axiom (funct Q), where Q is a basic role, or (3) a concept membership axiom A(a), where A € A and a € I, or (4) a role membership axiom P(a, c), where P € Ra and a, c € I.	[redacted]
A TBox is a finite set of concept inclusion and functionality axioms. An ABox is a finite set of concept and role membership axioms. A knowledge base KB = (T, A) consists of a TBox T and an ABox A. CQs and BCQs are defined as usual, with concept and role membership axioms as atoms (over variables and individuals as arguments).	[redacted]
The description logic DL-LiteR allows for (5) role inclusion axioms Q C R, rather than functionality axioms, where Q is a basic role, and R is a role.	[redacted]
Example 17 Consider the sets of atomic concepts, abstract roles, and individuals A, RA, and I, respec¬tively, given as follows: A = {Scientist, Article, ConferencePaper, JournalPaper}, Ra = {hasAuthor, hasFirstAuthor, isAuthorOf}, I = {i1,i2}-	[redacted]
The following concept inclusion axioms express that (i) conference and journal papers are articles, (ii) conference papers are not journal papers, (iii) every scientist has a publication, and (iv) isAuthorOf relates scientists and articles: (i) ConferencePaper C Article,	[redacted]

JournalPaper C Article, (ii) ConferencePaper C — JournalPaper, Scientist C BisAuthorOf, (iii) BisAuthorOfC Scientist, BisAuthorOf- C Article.	[redacted]
Some role inclusion and functionality axioms are as follows; they express that (v) isAuthorOf is the inverse of hasAuthor, and (vi) hasFirstAuthor is functional: (v) isAuthorOf- C hasAuthor, hasAuthor- C isAuthorOf, (vi) (funct hasFirstAuthor).	[redacted]
The following are some concept and role memberships, which express that the individual i1 is a scientist who authors the article i2: Scientist(i1), isAuthorOf(i1,i2), Article(i2).	[redacted]
7.2 Semantics of DL-LiteF and DL-LiteR	[redacted]
The semantics is defined via standard first-order interpretations. An interpretation 1 = (A , 1) consists of a nonempty (abstract) domain A1 and a mapping 1 that assigns to each atomic concept C € A a subset of A1, to each abstract role R € Ra a subset of A1 x A1, and to each individual a € I an element of A1. Here, different individuals are associated with different elements of A1 (unique name assumption). The mapping 1 is extended to all concepts and roles by: • (P-)1 = {(a,b) | (b,a) €P1}; • (—Q)1 = A1 x A1 - Q1; • (3Q)1 = {x € A1 |3y: (x,y) € Q1}; • (—B)1 = A1 \ B1.	[redacted]
The satisfaction of an axiom F in the interpretation 1 = (A1, 1), denoted 11= F, is defined as follows: (1) 11=	[redacted]

B C C iff B1 C C1; (2) 11= (funct Q) iff (o, o') € Q1 and (o, o") € Q1 implies o' = o"; (3) 1 |= A(a) iff a1 € A1; (4) 11= P(a, b) iff (a1, b1) € P1; and (5) 11= Q C R iff Q1 C R1. The interpretation 1 satisfies the axiom F, or 1 is a model of F, iff 11= F. The interpretation 1 satisfies a knowledge base KB = (T, A), or 1 is a model of KB, denoted 11= KB, iff 11= F for all F € TU A. We say that KB is satisfiable (resp., unsatisfiable) iff KB has a (resp., no) model. The semantics of CQs and BCQs is as usual in first-order logic.	[redacted]
7.3 Translation of DL-LiteF and DL-LiteR into Datalog± The translation T from the elementary ingredients and axioms of DL-LiteF and DL-LiteR into Datalog± is defined as follows:	[redacted]
(1) Every atomic concept A € A is associated with a unary predicate T(A) = PA € R, every abstract role P € Ra is associated with a binary predicate T(P) = pp € R, and every individual i € I is associated with a constant T(i) = Cj, € A.	[redacted]
(2) Every concept inclusion axiom B C C is translated to the TGD or constraint T(B C C)= T'(B) ^ T"(C), where	[redacted]
(i) T'(B) is defined as pA(X), pP(X, Y), and pP(Y, X), if B is of the form A, 3P, and 3P-, respectively, and	[redacted]
(ii) T"(C) is defined as PA(X), 3ZpP(X, Z), 3ZpP(Z, X), —PA(X), —pP(X, Y'), and —pP(Y', X), if C is of form A, 3P, 3P-, — A, —3P, and —3P-, respectively.	[redacted]
(3) The functionality axioms (funct P) and (funct P-) are under T translated to the EGDs pP(X, Y) A	[redacted]

PP(X, Y') ^ Y = Y' and PP(X, Y) A PP(X',Y) ^ X = X', respectively.

[redacted]

(4) Every concept membership axiom A(a) is under T translated to the database atom pA(ca), and every role membership axiom P(a, b) to the database atom pP(ca, cb).

[redacted]

(5) Every role inclusion axiom Q C R is translated to the TGD or constraint T(Q C R) = T'(Q) ^ T''(R) , where

[redacted]

(ii) t''(R) is defined as pp(X, Y), pp(Y, X), —pp(X, Y), and —pp(Y, X), if R is of the form P, P-, —P, and —P -, respectively.

[redacted]

Example 18 The concept inclusion axioms of Example 17 are translated to the following TGDs and constraints (where we identify atomic concepts and roles with their predicates):

ConferencePaper(X) — Article(X), JournalPaper(X) — Article(X), ConferencePaper(X) — — JournalPaper (X), Scientist(X) — BZ isAuthorOf(X, Z), isAuthorOf(X, Y) — Scientist(X), isAuthorOf(Y, X) — Article(X).

[redacted]

The role inclusion and functionality axioms of Example 17 are translated to the following TGDs and EGDs:

isAuthorOf(Y, X) — hasAuthor(X, Y), hasAuthor(Y, X) — isAuthorOf(X, Y), hasFirstAuthor(X, Y), hasFirstAuthor(X, Y') — Y = Y'.

The concept and role membership axioms of Example 17 are translated to the following database atoms (where we also identify individuals with their constants):

[redacted]

Scientist(i1), isAuthorOf(i1,i2), Article(i2).

Every knowledge base KB in DL-

[redacted]

Lite S, S € {F, R}, is then translated into a database DKB, set of TGDs £KB , and disjunction of queries QKB as follows: (i) the database DKB is the set of all t(0) such that 0 is a concept or role membership axiom in KB, (ii) the set of TGDs £KB is the set of all TGDs resulting from t (0) such that 0 is a concept or role inclusion axiom in KB, and (iii) QKB is the disjunction of all queries resulting from constraints and EGDs t(0) such that 0 is a concept inclusion, or role inclusion, or functionality axiom in KB (satisfying any query Q occurring in QKB means violating a constraint or EGD).	[redacted]
The following lemma shows that the TGDs generated from a DL-LiteR knowledge base are in fact lin¬ear TGDs, and that the TGDs and EGDs generated from a DL-LiteF knowledge base are in fact linear TGDs and NC keys, respectively. Here, the fact that the generated TGDs and EGDs are linear and keys, respec¬tively, is immediate by the above translation. Proving the NC property for the generated keys boils down to showing that keys resulting from functionality axioms (funct P) are NC with TGDs from concept inclusion axioms B C BP and B C BP-.	[redacted]
Lemma 16 Let KB be a knowledge base in DL-Lite S, S € {F, R}. Then, (a) every TGD in £KB is linear. If KB is in DL-LiteF, and £K is the set of all EGDs encoded in QKB, then (b) every EGD in £K is a key, and (c) £k is NC with £KB.	[redacted]
The next result shows that BCQs addressed to knowledge bases in DL-	[redacted]

LiteF and DL-LiteR can be re¬duced to BCQs in linear Datalog±. This important result follows from the above Lemma 16 and Theorem 14 (stating that the NC property for keys implies their separability relative to a set of TGDs). Here, recall that for disjunctions of BCQs Q, D U £ |= Q iff D U £ |= Q for some BCQ Q in Q.	[redacted]
Theorem 17 Let KB be a knowledge base in DL-LiteS, S € {F, R}, and let Q be a BCQ for KB. Then, Q is satisfied in KB iff DKB U £KB |= Q V QKB.	[redacted]
As an immediate consequence, the satisfiability of knowledge bases in DL-LiteF and DL-LiteR can be reduced to BCQs in Datalog±. Intuitively, the theorem follows from the observation that the unsatisfiability of KB is equivalent to the truth of ± in KB, which is in turn equivalent to DKB U £KB |= QKB .	[redacted]
Theorem 18 Let KB be a knowledge base in DL-LiteS, S € {F, R}. Then, KB is unsatisfiable iff DKB U £KB |= QKB. The next important result shows that Datalog± is strictly more expressive than both DL-LiteF and DL- LiteR. The main idea behind its proof is to show that neither DL-LiteF nor DL-LiteR can express the TGD p(X) ^ q(X,X).	[redacted]
Theorem 19 Datalog± is strictly more expressive than DL-LiteF and DL-LiteR.	[redacted]
8 Ontology Querying in DL-Lite A	[redacted]
We now generalize the results of Section 7 to the description logic DL-Lite A of the DL-Lite family [89]. We first recall the syntax and the semantics of DL-Lite A . We	[redacted]

then define the translation and the representation and expressivity results.	[redacted]
8.1 Syntax of DL-Lite A As for the elementary ingredients of DL-Lite A , let D be a finite set of atomic datatypes d, which are associ¬ated with pairwise disjoint sets of data values Vd. Let A, Ra, Rd, and I be pairwise disjoint sets of atomic concepts, atomic roles, atomic attributes, and individuals, respectively, and let V = (JdeD Vd.	[redacted]
Roles, concepts, attributes, and datatypes are defined as follows: • A basic role Q is either an atomic role P € Ra or its inverse P-. A (general) role R is either a basic role Q or the negation of a basic role —Q.	[redacted]
• A basic concept B is either an atomic concept A € A, or an existential restriction on a basic role Q, denoted 3Q, or the domain of an atomic attribute U € Rd, denoted 5(U). A (general) concept C is either the universal concept Tc, or a basic concept B, or the negation of a basic concept —B, or an existential restriction on a basic role Q of form 3Q.C, where C is a concept.	[redacted]
• A (general) attribute V is either an atomic attribute U € Rd or the negation of an atomic attribute —U.	[redacted]
• A basic datatype E is the range of an atomic attribute U € Rd, denoted p(U). A (general) datatype F is either the universal datatype TD or an atomic datatype d € D.	[redacted]
An axiom has one of the following forms: (1) B C C (concept inclusion axiom), where B is a basic concept, and C is a concept; (2) Q C R (role	[redacted]

inclusion axiom), where Q is a basic role, and R is a role; (3) U C V (attribute inclusion axiom), where U is an atomic attribute, and V is an attribute; (4) E C F (datatype inclusion axiom), where E is a basic datatype, and F is a datatype; (5) (funct Q) (role functionality axiom), where Q is a basic role; (6) (funct U) (attribute functionality axiom), where U is an atomic attribute; (7) A(a) (concept membership axiom), where A is an atomic concept and a € I; (8) P(a, b) (role membership axiom), where P is an atomic role and a,b € I; and (9) U(a, v) (attribute membership axiom), where U is an atomic attribute, a € I, and v € V.	[redacted]
We next define knowledge bases, which consist of a restricted finite set of inclusion and functionality axioms, called TBox, and a finite set of membership axioms, called ABox. We first define the restriction on inclusion and functionality axioms. A basic role P or P- (resp., an atomic attribute U) is an identifying property in a set of axioms S iff S contains a functionality axiom (funct P) or (funct P-) (resp., (funct U)). Given an inclusion axiom a of the form X C Y (resp., X C —Y), a basic role (resp., atomic attribute) Y appears positively (resp., negatively) in its right-hand side of a. A basic role (resp., atomic attribute) is primitive in S iff it does not appear positively in the right-hand side of an inclusion axiom in S and it does not appear in an expression BQ.C in S. We can now define knowledge bases. A	[redacted]

TBox is a finite set T of inclusion and functionality axioms such that every identifying property in T is primitive. Intuitively, identifying properties cannot be specialized in T, i.e., they cannot appear positively in the right-hand side of inclusion axioms in T. An ABox A is a finite set of membership axioms. A knowledge base KB = (T, A) consists of a TBox T and an ABox A. As usual, CQs and BCQs use concept and role membership axioms as atoms (over variables and individuals).

8.2 Semantics of DL-LiteA

The semantics of DL-Lite A is defined in terms of standard typed first-order interpretations. An interpretation 1 = (A1, 1) consists of (i) a nonempty domain A1 = (AQ, AV), which is the disjoint union of the domain of objects AO and the domain of values AV = (J deD A1, where the A1 's are pairwise disjoint domains of values for the datatypes d € D, and (ii) a mapping 1 that assigns to each datatype d € D its domain of values A^, to each data value v € Vd an element of A1 (such that v = w implies v1 = w1), to each atomic concept A € A a subset of AQ, to each atomic role P € Ra a subset of AQ x AQ , to each atomic attribute P € Rd a subset of AQ x AV, to each individual a € I an element of AQ (such that a = b implies a1 = b1). Note that different data values (resp., individuals) are associated with different elements of AV (resp., AQ) (unique name assumption). The extension of 1 to all concepts, roles, attributes, and datatypes, and the satisfaction of an axiom a in 1 = (A1,

[redacted]

[redacted]

[redacted]

1), denoted 11= a, are defined by: • (Td)x = AV and (Tc)x = AQ; • (—U)x = (AQ x AV) - U1; • (—Q)1 = (AQ x AQ) - Q1; • (P(U))X = {v € AV | Bo:(o,v) € U1}; • (^(U))x = {o € AQ | Bv: (o, v) € U1}; • (p-)1 ={(o o') € AO x AO| (^o) € p1}; • (BQ)1 = {o € AQ | Bo': (o, o') € Q1}; • (BQ.C)x = {o € AQ | Bo': (o, o') € Q1, o' € C1}; • (—B)1 = AO \ B1.	[redacted]
The satisfaction of an axiom a in the interpretation 1 = (A1, 1), denoted 11= a, is defined as follows:	[redacted]
(1) 11= B C C iff B1 C C1, (2) 11= Q C R iff Q1 C R1, (3) 11= E C F iff E1 C F1, (4) 11= U C V iff U1 C V1, (5) 11= (funct Q) iff (o, q), (o, q') € Q1 implies q = q', (6) 11= (funct U) iff (o, v), (o, v') € U1 implies v = v', (7) 11= A(a) iff a1 € A1, (8) 11= P(a, b) iff (a1, b1) € P1, (9) 11= U(a, v) iff (a1, v1) € U1. The interpretation 1 satisfies the axiom a, or 1 is a model of a, iff 11= a. We say 1 satisfies a knowledge base KB = (T, A), or 1 is a model of KB, denoted 11= KB, iff 11= a for all a € TU A. We say KB is satisfiable (resp., unsatisfiable) iff KB has a (resp., no) model. The semantics of CQs and BCQs is as usual in first-order logic.	[redacted]
8.3 Translation of DL-Lite A into Datalog±	[redacted]
The translation T from the elementary ingredients and axioms of DL-Lite A into Datalog± is defined as follows:	[redacted]

(1) Every data value v has a constant T(v) = cv € A such that the T(Vd)’s for all datatypes d € D are pairwise disjoint. Every datatype d € D has under T a predicate T(d) = pd along with the constraint Pd(X) A Pd (X) ^ ± for all pairwise distinct d, d' € D. Every atomic concept A € A has a unary predicate T (A) = PA € R, every abstract role P € Ra has a binary predicate T (P) = pP € R, every attribute U € Rd has a binary predicate T(U) = pu € R, and every individual i € I has a constant T(i) = ci € A -UdgD T(Vd).	[redacted]
(2) Every concept inclusion axiom B C C is translated to the TGD or constraint T(B C C)= T'(B) ^ T"(C), where	[redacted]
(i) T'(B) is defined as pA(X), pP(X, Y), pP(Y,X), and pu(X, Y), if B is of the form A, 3P, 3P-, and 5(U), respectively, and	[redacted]
(ii) T"(C) is defined as PA(X), 3ZpP(X, Z), 3ZpP(Z, X), 3Zpu(X, Z), —PA(X), — pP(X, Y'), —pP(Y', X), —pu(X, Y'), 3ZpP(X, Z) A PA(Z), and 3ZpP(Z, X) A PA(Z), if C is of form A, 3P, 3P-, 5(U), —A, —3P, —3P-, — 5(U), 3P.A, and 3P-.A, respectively.	[redacted]
Note that concept inclusion axioms B C Tc can be safely ignored, and concept inclusion axioms B C 3Q.C can be expressed by the two concept inclusion axioms B C 3Q.A and A C C, where A is a fresh atomic concept. Note also that the TGDs with two atoms in their heads abbreviate their equivalent sets of TGDs with singleton atoms in the heads.	[redacted]
(3) The functionality axioms (funct P) and (funct P-) are under T translated to the EGDs pP(X, Y) A	[redacted]

pP (X, Y') ^ Y = Y' and pP (X, Y) A pP (X', Y) ^ X = X', respectively. The functionality axiom (funct U) is under T translated to the EGD pu(X, Y) A pu(X, Y') ^ Y = Y'.	[redacted]
(4) Every concept membership axiom A(a) is under T translated to the database atom pA(ca). Every role membership axiom P(a, b) is under T translated to the database atom pP(ca, cb). Every attribute membership axiom U(a, v) is under T translated to the database atom pu(ca, cv).	[redacted]
(5) Every role inclusion axiom Q C R is translated to the TGD or constraint T(Q C R) = T'(Q) ^ T"(R) , where	[redacted]
(i) T'(Q) is defined as pP(X, Y) and pP(Y, X), if Q is of the form P and P-, respectively, and	[redacted]
(ii) T"(R) is defined as PP(X, Y), PP(Y, X), — PP(X, Y), and — PP(Y, X), if R is of the form P, P-, —P, and — P-, respectively.	[redacted]
(6) Attribute inclusion axioms U C U' and U C —U' are under T translated to the TGD pu (X, Y) ^ pjji (X, Y) and the constraint pu(X, Y) ^ — pu> (X, Y), respectively.	[redacted]
(7) Every datatype inclusion axiom p(U) C d is under T translated to the TGD pjj(Y, X) ^ pd(X). Note that datatype inclusion axioms p(U) C TD can be safely ignored.	[redacted]
Every knowledge base KB in DL-Lite A is then translated into a database DKB, set of TGDs £KB, and disjunction of queries QKB as follows: (i) DKB is the set of all T(4>) such that 0 is a membership axiom in KB along with “type declarations” pd(v) for all their data values; (ii) £KB is the set of all	[redacted]

TGDs resulting from t(0) such that 0 is an inclusion axiom in KB; and (iii) QKB is the disjunction of all queries resulting from datatype constraints and from constraints and EGDs t (0) such that 0 is an inclusion or functionality axiom in KB.

[redacted]

The following result shows that Lemma 16 carries over to DL-Litea. That is, the TGDs and EGDs generated from a DL-Lite A knowledge base are in fact linear TGDs and NC keys, respectively. This follows from the observation that the new TGDs for DL-Lite A are also linear or equivalent to collections of linear TGDs, and that the keys are also NC with the new TGDs, due to the restricting assumption that all identifying properties in DL-Lite A knowledge bases are primitive.

[redacted]

Lemma 20 Let KB be a knowledge base in DL-Lite A, and let £K be the set of all EGDs encoded in QKB. Then, (a) every TGD in £KB is linear, (b) every EGD in £K is a key, and (c) £K is NC with £KB.

[redacted]

Consequently, also Theorem 17 carries over to DL-Lite a. That is, BCQs addressed to knowledge bases in DL-Lite A can be reduced to BCQs in Datalog±. Note that here and in the theorem below, we assume that every datatype has an infinite number of data values that do not occur in KB. Here, recall that for disjunctions of BCQs Q, D U £ |= Q iff D U £ |= Q for some BCQ Q in Q.

[redacted]

Theorem 21 Let KB be a knowledge base in DL-Lite A, and let Q be a BCQ for KB. Then, Q is satisfied in KB iff Dkb U £kb |= Q V Qkb.

[redacted]

Similarly, the satisfiability of

[redacted]

knowledge bases in DL-Lite A can be reduced to BCQs in Datalog±. This result is formally expressed by the following theorem, which is an extension of Theorem 18 to DL-Lite a.	[redacted]
Theorem 22 Let KB be a knowledge base in DL-Lite A. Then, KB is unsatisfiable iff DKB U £KB |= QKB.	[redacted]
Finally, Datalog± is also strictly more expressive than DL-Lite a, which is formulated by the next theo¬rem, extending Theorem 19 to DL-Lite a.	[redacted]
Theorem 23 Datalog± is strictly more expressive than DL-Lite A.	[redacted]
9 Ontology Querying in Other Description Logics	
In this section, we show that also the other tractable description logics of the DL-Lite family can be reduced to Datalog±. We also recall that F-Logic Lite is a special case of weakly-guarded Datalog±. Furthermore, we show that the tractable description logics EL and EL1f can be reduced to guarded Datalog± and to guarded Datalog± with non-conflicting keys, respectively.	[redacted]
9.1 Ontology Querying in the DL-Lite Family	[redacted]
A complete picture of the DL-Lite family of description logics (with binary roles) [34] is given in Fig. 4; note that the arrows with filled lines represent proper generalizations, while the ones with dashed lines en¬code generalizations along with syntactical restrictions. In addition to DL-LiteF, DL-LiteR, and DL-Lite a, the DL-Lite family consists of further description logics, namely, (i)	[redacted]

DL-Litecore, which is the intersection of DL-LiteF and DL-LiteR, (ii) DL-LiteA, which is obtained from DL-Lite A by adding role attributes and iden¬tification constraints, and (iii) DL-LiteF,n, DL-LiteR,n, and DL-LiteA n, which are obtained from DL-LiteF,	[redacted]
Figure 4: The DL-Lite family of description logics.	[redacted]
DL-LiteR, and DL-LiteA, respectively, by additionally allowing conjunctions in the left-hand sides of inclu¬sion axioms (without increase of complexity, which is related to the addition of Boolean role constructors in some popular description logics, as explored in [97]). Furthermore, each above description logic (with binary roles) DL-Lite X has a variant, denoted DLR-LiteX, which additionally allows for n-ary relations, along with suitable constructs to deal with them.	[redacted]
Clearly, since DL-Litecore is a restriction of DL-LiteF, it can also be reduced to Datalog±. In the fol¬lowing, we show that all the other description logics of the DL-Lite family can similarly be reduced to Datalog±.	[redacted]
A role attribute UR [32] denotes a binary relation between objects and values. Role attributes come along with: (1) introducing a set of atomic role attributes, (2) extending (general) concepts by expressions of the form 5F(U), 35F(UR), and 35F(UR)-, denoting the set of all objects, object pairs, and inverses of object pairs, respectively, that an atomic (concept) attribute U or an atomic role attribute UR relates to	[redacted]

values of a (general) datatype F, (3) extending basic datatypes by atomic datatypes d and the expression p(UR), denoting the range of the atomic role attribute Ur, (4) extending (general) datatypes by basic datatypes E and their negations —E, (5) adding (general) role attributes VR, which are either atomic role attributes UR or their negations —UR, (6) extending basic roles by the expressions 5(UR) and 5(UR)-, denoting the set of all object pairs and inverses of object pairs, respectively, that an atomic role attribute UR relates to values, (7) extending (general) roles by the expressions 5F(UR) and 5F(UR)-, denoting the set of all object pairs and inverses of object pairs, respectively, that an atomic role attribute UR relates to values of a general datatype F, (8) adding role attribute inclusion axioms UR C VR and functionality axioms (funct Ur), where UR (resp., VR) is an atomic (resp., a general) role attribute, (9) adding membership axioms UR(a, b, c) for atomic role attributes Ur, and (10) extending the notion of identifying property to also include all atomic role attributes in functionality axioms. Note that all axioms with concepts 3Q.C and with concepts and roles containing the operator 5F can be reduced to other axioms without them [32]. The translation T of DL-LiteA into Datalog± is then extended to the remaining axioms by:

BZ, Z'puR(Z,X, Z'), —puR(X, Z, Z'), and —puR(Z,X, Z') if C is of the form B5(Ur), B6(Ur)-, —B£(UR), and —B£(UR)-, respectively;

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

(2) functionality axioms (funct UR) are under T translated to the EGD pUR (X, Y, Z) ApUR (X, Y, Z') ^ Z = Z';	[redacted]
(3) role attribute membership axioms UR(a, b, c) are under T translated to the database atom pUR (ca cb, cc);	[redacted]
(4) for role inclusion axioms Q C R, we additionally define (i) T'(Q) as pUR (X, Y, Y') and pUR (Y, X, Y') if Q is of the form £(UR) and £(UR)-, respectively, and (ii) T''(R) as BZpUR(X, Y, Z), BZpUR(Y, X, Z), —pUR(X, Y, Z), and — pUR(Y, X, Z) if Rhas the form £(UR), £(UR)-, — £(UR), and — £(UR)-, respectively;	[redacted]
(5) role attribute inclusion axioms UR C UR and UR C —UR are under T translated to the TGD pUR (X, Y, Z) ^ p^R (X, Y, Z) and the constraint pUR (X, Y, Z) ^ — p^R (X, Y, Z), respectively; and	[redacted]
(6) datatype inclusion axioms E C F are under t translated to t'(E) ^ t''(F), where (i) t'(E) is de¬fined as pd(X) and pUR(Y, Y',X), if E is of form d and p(UR), respectively, and (ii) t''(F) as pd(X), puR(Z, Z',X), —pd(X), and —puR(Z, Z',X), if F is of form d, p(Ur), —d, and —p(Ur), respectively.	[redacted]
An identification axiom [33] is of the form (id B Ii,..., In), with n ^ 1, where B is a basic concept, and each Ij, j € {1,..., n}, is either an atomic attribute or a basic role. Such an axiom encodes that the combination of properties Ii,..., In identifies the instances of the basic concept B. The notion of identify¬ing property is then extended to also include all atomic attributes and basic roles that	[redacted]

occur in identification axioms. The translation t of DL-Lite A into Datalog± is extended by mapping each such axiom under t to the EGD (which is a slight extension of Datalog± to also include Ii,..., In as a key of a virtual relation R(B, Ii,..., In)):	[redacted]
TB(X) A An=i Tii(X, Yi) A TB(X') A An=i Tii(X', Y,) ^ X = X',	
where (i) tb(X) is defined as pa(X), pp (X, Y), pp (Y, X), puR (X, Y, Y'), puR (Y, X, Y'), and pu(X, Y), if B is of form A, BP, BP-, B£(UR), B£(UR)-, and 5(U), respectively, and (ii) t/(X, Y) is p^(X, Y), pP(X, Y), pP(Y, X), pUR(X, Y, Y'), and pUR(Y, X, Y'), if I is of form U, P, P-, 6(UR), and 6(UR)-, respectively.	[redacted]
The following result shows that Theorems 21 and 22 carry over to DL-LiteA, i.e., both BCQs addressed to knowledge bases KB in DL-LiteA as well as the satisfiability of such KB can be reduced to BCQs in Datalog±. Here and in the following, the database DKB, the set of TGDs £KB, and the disjunction of queries QKB are defined as in Sections 7 and 8, except that we now use the corresponding extended translation t , rather than those of Sections 7 and 8, respectively.	[redacted]
Theorem 24 Let KB be a knowledge base in DL-LiteA, and let Q be a BCQ for KB. Then, (a) Q is satisfied in KB iff DKB U £KB |= Q V QKB, and (b) KB is unsatisfiable iff DKB U £KB |= QKB.	[redacted]
The three description logics DL-LiteF,n, DL-LiteR,n, and DL-LiteA n are obtained from DL-LiteF, DL-LiteR, and DL-LiteA, respectively, by additionally allowing	[redacted]

conjunctions in the left-hand sides of inclusion axioms. These DLs can be encoded by Datalog± with multi-linear TGDs. To this end, the translation t of DL-LiteF, DL-LiteR, and DL-LiteA into Datalog± is extended by first mapping the left-hand sides of inclusion axioms to the conjunctions of their previous mappings under T. Every body atom p(X, Y) in a TGD a with variables Y that do not occur in the head of a is then replaced by a new body atom p' (X), where p is a fresh predicate, along with adding the TGD p(X, Y) ^ p'(X).	[redacted]
The next result shows that Theorems 17, 18, and 24 carry over to DL-LiteF,n, DL-LiteR,n, and DL- LiteA n, i.e., both BCQs addressed to knowledge bases KB in these DLs and the satisfiability of such KB are reducible to BCQs in Datalog±.	[redacted]
Theorem 25 Let KB be a knowledge base in DL-LiteF,n, DL-LiteR,n, or DL-LiteA n, and let Q be a BCQ for KB. Then, (a) Q is satisfied in KB iff DKB U |= Q V QKB, and (b) KB is unsatisfiable iff DKB u £KB = QKB .	[redacted]
For each of the above description logics (with binary roles) DL-LiteX, the description logic DLR-LiteX is obtained from DL-LiteX by additionally allowing for n-ary relations, along with suitable constructs to deal with them. More concretely, (1) the construct 3P in basic concepts is generalized to the construct 3i: R, where R is an n-ary relation and i € {1,, n}, which denotes the projection of R on its i-th component; (2) the construct 3Q.C in (general) concepts is generalized to	[redacted]

the construct 3i: R.Ci... Cn, where R is an n¬ary relation, the Ci's are (general) concepts, and i € {1,..., n}, which denotes those objects that participate as i-th component to tuples of R where the j-th component is an instance of Cj, for all j € {1,..., n}; (3) one additionally allows for functionality axioms (funct I: R), stating the functionality of the i-th component of R; and (4) one additionally allows for inclusion axioms between projections of relations R1[i1,... ,ik] C R2[j1, ...,jk], where R1 is an n-ary relation, i1,... ,ik € {1,..., n}, ip = iq if p = q, R2 is an m-ary relation, j1,... ,jk € {1,..., m}, and jp = jq if p = q. Since the construct 3i: R.C1 ...Cn can be removed in a similar way as 3Q.C in the binary case [32], it only remains to define the translation t into Datalog± for the following cases:	[redacted] [redacted]
(1) for concept inclusion axioms B C C, we additionally define (i) t'(B) as pR(Y1,..., Yi-1,X, Yi+1, ..., Yn), if B = 3i: R, and (ii) t"(C) as 3Zb ..., Zi-1, Zi+1,..., Zn pr(Z1, ..., Zi-1, X, Zi+1, ..., Zn) and —pR(Y1,..., Y—1, X, Y[+1,..., Yn), if B is of the form 3i: R and —3i: R, respectively;	[redacted]
(2) every functionality axiom (funct i: R) is under t mapped to the set of all EGDs pR(Y1,..., Yi-1, X, Yi+1,..., Yn) A pr (Y1,..., Yi-1, X, Yi+1, -,YI) ^ Yj = Yj such that j € {1,..., n} and j = i; and	[redacted]
(3) every inclusion axiom R1[i1,..., ik] C R2[j1,..., jk] is under t mapped to the TGD pRl (X) ^ 3Zpr2(Z'), where Zj = Xit for all l € {1,..., k}, and Z is the vector of all variables Zj with j €{1,...,m}-	[redacted]

{j1,...,jk }.

The next theorem finally shows that Theorem 25 carries over to DLR-LiteF,n, DLR-LiteR,n, and DLR-Lite+A,n (and so also to all less expressive n-ary description logics of the DL-Lite family), i.e., both BCQs addressed to knowledge bases KB in these DLs and the satisfiability of such KB are reducible to BCQs in Datalog±.

Theorem 26 Let KB be a knowledge base in DLR-LiteF,n, DLR-LiteR,n, or DLR-LiteA n, and let Q be a BCQ for KB. Then, (a) Q is satisfied in KB iff DKB U £KB |= Q V QKB, and (b) KB is unsatisfiable iff

DKB u £KB |= QKB .

Finally, observe also that Datalog± is strictly more expressive than every description logic of the DL- Lite family and its extension with n-ary relations, which follows from the next theorem, generalizing Theo¬rems 19 and 23.

Theorem 27 Datalog± is strictly more expressive than DL-LiteF,n, DL-LiteR,n, DL-LiteA n, DLR-LiteF,n, DLR-LiteR,n, and DLR-LiteA n.

9.2 Ontology Querying in F-Logic Lite, EL, and ELIf

Other ontology languages that are reducible to Datalog± include F-Logic Lite [29], which is a special case of weakly-guarded Datalog± [23]. F-Logic Lite is a limited version of F-Logic [59], which is a well-known object-oriented formalism; an F-Logic Lite schema can be represented as a fixed set of TGDs using meta¬predicates and a set of ground facts. In the rest of this section, we show that the tractable

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

description logics EL [11] and ELIf (which is the extension of EL by inverse and functional roles) [62] can be reduced to guarded Datalog± and to guarded Datalog± with non-conflicting keys, respectively.	[redacted]
The description logic EL has the following ingredients. We assume pairwise disjoint sets of atomic concepts, abstract roles, and individuals A, Ra, and I, respectively. A concept is either the top concept T, an atomic concept A, an existential role restriction BR.C, or a conjunction C n D, where R is an abstract role, and C and D are concepts. A TBox is a finite set of concept inclusion axioms C C D, where C and D are concepts, while an ABox is a finite set of concept and role membership axioms A(c) and R(c, d), respectively, where A is an atomic concept, R is an abstract role, and c and d are individuals. A knowledge base KB = (T, A) consists of a TBox T and an ABox A.	[redacted] [redacted]
We define a translation t from EL to Datalog± with guarded TGDs as follows. Atomic concepts, abstract roles, and individuals are translated under t in the same way as for DL-Lite. The same applies to concept and role membership axioms, which produce the database DKB for a given knowledge base KB in EL. As for concept inclusion axioms C C D, we can w.l.o.g. assume that C contains at most one existential role restriction BR.E, since any other existential role restriction can be replaced by a fresh atomic concept B along with the concept inclusion axiom BR.E C B. We then inductively define tx , where X is a	[redacted]

variable, for all concepts by tx(T) = T (i.e., logical truth), tx(A) = pa(X), tx(BR.C) = BZ (pR(X, Z) Atz(C)), and tx(C n D) = tx(C) A t'(tx(D)), where t' is a renaming of existentially quantified variables such that tx (C) and t'(tx (D)) have no such variables in common anymore. We finally define the translation t for all concept inclusion axioms by t(C C D) = tX(C) ^ tX(D), where tX(C) (resp., tX(D)) is obtained from tx(C) (resp., tx(D)) by a renaming of existentially quantified variables to eliminate common ones and by removing (resp., “moving out”) all existential quantifiers. We denote by £KB the resulting set of rules for KB. It is not difficult to verify that £KB is in fact a finite set of guarded TGDs.

[redacted]

The following immediate result finally shows that the tractable description logic EL can be reduced to guarded Datalog±, i.e., BCQs addressed to knowledge bases in EL can be reduced to BCQs in Datalog± with guarded TGDs.

[redacted]

Theorem 28 Let KB be a knowledge base in EL, and let Q be a BCQ for KB. Then, Q is satisfied in KB iff Dkb U £kb |= Q.

[redacted]

As for the description logic ELIf, rather than only abstract roles in concepts, we may have abstract roles and inverses of abstract roles, and we may additionally state that abstract roles and inverses of abstract roles are functional. The above translation T from EL to guarded Datalog± can be easily extended to also allow for inverses of abstract roles by defining additionally TX(BR- .C) = BZ (pR(Z, X) A TZ(C)), while the functionality of abstract roles and of

[redacted]

inverses of abstract roles can be encoded as EGDs that are unary keys and NC with the TGDS. TO verify the NC property, observe that we can w.l.o.g. assume that each concept D in concept inclusion axioms C C D is either an atomic concept A or of the form BS.B, where S is an abstract role or the inverse of an abstract role, and B is an atomic concept, since conjunctions in D can be removed by encoding C C Di n D2 as C C Di and C C D2, and since existential role restrictions BS.C with non-atomic concepts C in D can be replaced by existential role restrictions BS.B along with the concept inclusion axiom B C C, where B is a fresh atomic concept.

Note that guarded Datalog± and guarded Datalog± with non-conflicting keys are also strictly more ex-pressive than EL and ELIf, respectively, since they allow for predicates of arbitrary arity in rules (provided that guardedness holds).

10 Stratified Negation

In this section, we extend Datalog± by stratified negation. We first define the syntax of TGDs with negations in their bodies (called normal TGDs) and of BCQs with negations (called normal BCQs), and we introduce a canonical model semantics via iterative chases. We then show that answering safe normal BCQs from databases under stratified sets of guarded normal TGDs can be done on finite portions of these chases; as a consequence, it is data-tractable (resp., FO-rewritable) in the guarded (resp., linear) case. We finally define a perfect model semantics, and we show that it coincides with the

[redacted]

[redacted]

[redacted]

canonical model semantics (which also implies that the canonical model semantics is independent of a concrete stratification), and that it is an isomorphic image of the perfect model semantics of a corresponding normal logic program with function symbols. The fact that the canonical model semantics coincides with a perfect model semantics, the independence of the canonical model semantics of a concrete stratification, and the fact that the perfect model semantics is an isomorphic image of the perfect model semantics of a corresponding normal logic program show that we provide a natural stratified negation for query answering over ontologies, which has been an open problem in the DL community to date, since it is in general based on several strata of infinite models. By the results of Sections 5 and 6 (which can easily be extended to stratified sets of normal TGDs), and of Sections 7 to 9, this also provides a natural stratified negation for the DL-Lite family. 10.1 Normal TGDs and BCQs We first introduce the syntax of normal TGDs, which are informally TGDs that may also contain negated atoms in their bodies. Given a relational schema R,a normal TGD (NTGD) has the form VXVY $(X, Y) ^ BZ ^(X, Z), where $(X, Y) is a conjunction of atoms and negated atoms over R, and ^(X, Z) is a con¬junction of atoms over R (all atoms without nulls). It is also abbreviated as $(X, Y) ^ BZ ^(X, Z). As in the case of standard TGDs, we can assume that ^(X, Z) is a singleton atom. We denote by	[redacted]

head(ct) the atom in the head of ct, and by body+ (ct) and body-(ct) the sets of all positive and negative (“—”-free) atoms in the body of ct, respectively. We say that ct is guarded iff it contains a positive atom in its body that contains all universally quantified variables of ct. For ease of presentation, guarded NTGDs contain no constants. We say that ct is linear iff ct is guarded and has exactly one positive atom in its body.	[redacted]
As for the semantics of normal TGDs ct, we say that ct is satisfied in a database D for R iff, whenever there exists a homomorphism h for all the variables and data constants in the body of ct that maps (i) all atoms of body+(ct) to atoms of D and (ii) no atom of body-(ct) to atoms of D, then there exists an extension h' of h that maps all atoms of head (ct) to atoms of D.	[redacted]
We next add negation to BCQs as follows. A normal Boolean conjunctive query (NBCQ) Q is an existentially closed conjunction of atoms and negated atoms (without nulls) of the form 3Xp1(X) A ■ ■ ■ A pm(X) A —pTO+1(X) A ■ ■ ■ A —pm+n(X),	[redacted]
where m ^ 1, n ^ 0, and the variables of the pi’s are among X. We denote by Q+ (resp., Q-) the set of all positive (resp., negative (“—”-free)) atoms of Q. We say Q is safe iff every variable in a negative atom in Q also occurs in a positive atom in Q.	[redacted]
Example 19 Consider the following set of guarded normal TGDs £, expressing that (1) if a driver has a non-valid license and drives, then he	[redacted]

violates a traffic law, and (2) a license that is not suspended is valid: a: hasLic(D, L), drives(D), — valid(L) ^ 3I viol(D, I); a': hasLic(D,L), —susp(L) ^ valid,(L).	[redacted]
Then, asking whether John commits a traffic violation and whether there exist traffic violations without driv¬ing can be expressed by the two safe normal BCQs Q1 = 3X viol(john, X) and Q2 = 3D, I viol(D, I) A —drives(D), respectively.	[redacted]
10.2 Canonical Model Semantics We now define the concept of a stratification for normal TGDs as well as the canonical model semantics of databases under stratified sets of guarded normal TGDs via iterative chases along a stratification. We then provide several semantic results around canonical models, and we finally define the semantics of safe normal BCQs via such canonical models.	[redacted]
We define the notion of stratification for normal TGDs by generalizing the classical notion of stratifica¬tion for Datalog with negation but without existentially quantified variables [7] as follows. A stratification of a set of normal TGDs £ is a mapping ^: R ^ {0,1, ... ,k} such that for each normal TGD a € £:	[redacted]
(i) ^(pred(head(a))) ^ ^(pred(a)) for all a € body+(a); (ii) ^(pred(head(a))) > ^(pred(a)) for all a € body-(a). We call k ^ 0 the length of ^. For every i € {0,..., k}, we then define Di = {a € D | ^(pred(a)) = i} and D* = {a € D | ^(pred(a)) ^ i}, as well as £ = {a € £ | ^(pred(head(a))) = i} and £* = {a € £ | ^(pred(head(a))) ^ i}. We say that £ is stratified iff it has a	[redacted]

stratification ^ of some length k ^ 0.

Example 20 Consider again the set of guarded normal TGDs £ of Example 19. It is then not difficult to verify that the mapping ^ where ^(susp) = ^(hasLic) = ^(drives) = 0, ^(valid) = 1, and ^(viol) = 2 is a stratification of £ of length 2. Hence, £ is stratified, and we obtain £o = 0, £1 = {a'}, and £2 = {a}.

We next define the notion of indefinite grounding, which extends the standard grounding (where rules are replaced by all their possible instances over constants) towards existentially quantified variables. A subset of the set of nulls AN is partitioned into infinite sets of nulls Aa,z (which can be seen as Skolem terms by which Z can be replaced), one for every a € £ (where £ is a set of guarded normal TGDs) and every existentially quantified variable Z in a. An indefinite instance of a normal TGD a is obtained from a by replacing every universally quantified variable by an element from A U AN and every existentially quantified variable Z by an element from Aa,z. The indefinite grounding of £, denoted ground(£), is the set of all its indefinite instances. We denote by HBs the set of all atoms built from predicates from £ and arguments from A U AN. We naturally extend the oblivious chase of D and £ to databases D with nulls, which are treated as new data constants; similarly, normal TGDs are naturally extended by nulls.

We are now ready to define canonical models of databases under stratified sets of guarded normal TGDs via iterative chases as follows.

[redacted]

[redacted]

[redacted]

Note that this is slightly different from [24], where we define the canonical model semantics as iterative universal models. The main reason why we use the slightly stronger definition here is that it makes canonical models isomorphic to canonical models of a corresponding normal logic program with function symbols (cf. Corollary 38).

[redacted]

Definition 5 Let R be a relational schema. Given a database D for R under a stratified set of guarded normal TGDs £ on R, we define the sets S, along a stratification ^: R ^ {0,1, ..., k} of £ as follows:

[redacted]

(i) So = chase(D, £0);

(ii) if i > 0, then S, — chase (S,—i, £^'), where the set of TGDs £^' is obtained fro^fr ground(£,) by (i) deleting all ct such that body-(ct) n Si—i = 0 and (ii) removing the negative body from the remaining ct’s.

[redacted]

Then, Sk is a canonical model of D and £.

[redacted]

Example 21 Consider again the set of guarded normal TGDs £ of Example 19 and the database D = {susp(l), drives(john, c), hasLic(john, l)}. Since £0 = 0, £i = {ct'}, and £2 = {ct} (seeExample 20), we obtain S0 = Si = D, and S2 is isomorphic to D U {viol(john, i)}, where i is a null.

[redacted]

Observe that, given a database D and a set of guarded TGDs £, the oblivious chase of D and £ min¬imizes equality between newly introduced nulls, but every TGD CT = $(X, Y) ^ BZ ^(X, Z) generates exactly one new null for every indefinite ground instance of the

[redacted]

body of CT satisfied in the oblivious chase. The following theorem shows that this policy is actually closely related to the least Herbrand model seman¬tics of positive logic programs with function symbols: there exists an isomorphism from the oblivious chase of D and £ to the least Herbrand model of a corresponding positive logic programs with function symbols. This provides a strong justification for using the oblivious chase in the above canonical model semantics of databases under stratified sets of guarded normal TGDS.

Given a set of guarded TGDs £, the functional transformation of £, denoted £f, is obtained from £ by replacing each TGD ct = $(X, Y) ^ BZ ^(X, Z) in £ by the generalized TGD CTf = $(X, Y) ^ ^(X, f<r(X, Y)), where fCT is a vector of function symbols /CT;Z for ct, one for every variable Z in Z. Observe that CTf now contains function symbols, but no existential quantifiers anymore. Furthermore, for every database D, it holds that D U £f is a positive logic program with function symbols, which has a canonical semantics via unique least Herbrand models. The notions of databases, queries, and models are naturally extended by such function symbols. An additional restriction on the notion of isomorphism is that nulls c € Act Z are associated with terms of the form /CT;Z(x, y).

Theorem 29 Let R be a relational schema, D be a database for R, and £ be a set of guarded TGDs on R. Then, there exists an isomorphism from chase (D, £) to the least

[redacted]

[redacted]

[redacted]

[redacted]

Herbrand model M of D and £f.

The following result shows that canonical models of databases D under stratified sets of guarded normal TGDs £ are in fact also models of D and £, which is a minimal property expected from the notion of “canonical model”. The proof is done by induction along a stratification of £, showing that every S, is a model of D and £*, using the chase construction of every S,. Thus, in particular, the canonical model Sk of D and £ is a model of D and £ = £.

[redacted]

Proposition 30 Let R be a relational schema, D be a database for R, and £ be a stratified set of guarded normal TGDs on R. Let S be a canonical model of D and £. Then, S is also a model of D and £.

[redacted]

In general, there are several canonical models of databases D under stratified sets of guarded normal TGDs £. The next result shows that they are all isomorphic. It is proved by induction along a stratification of £, showing that for any two constructions of canonical models S0,... ,Sk and T0,... ,Tk, it holds that every Si, is isomorphic to Ti, using the chase construction of every Si, and Ti. Thus, in particular, the two canonical models Sk and Tk of D and £ are isomorphic.

[redacted]

Proposition 31 Let R be a relational schema, D be a database for R, and £ be a stratified set of guarded normal TGDs on R. Let U and V be two canonical models of D and £. Then, U is isomorphic to V.

[redacted]

We finally define the semantics of safe normal BCQs addressed to databases D under stratified sets of guarded normal TGDs £ via their

[redacted]

canonical models as follows. A BCQ Q evaluates to true in D and £, denoted D U £ \=strat Q, iff there exists a homomorphism that maps Q to a canonical model Sk of D and £. A safe normal BCQ Q evaluates to true in D and £, denoted D U £ l=strat Q, iff there exists a homomorphism from Q+ to a canonical model of D and £, which cannot be extended to a homomorphism from some Q+ U {a}, where a € Q-, to the canonical model of D and £. Note that the fact that every canonical model of D and £ is isomorphic to all other canonical models of D and £ (cf. Proposition 31) assures that the above definition of the semantics of safe normal BCQs does not depend on the chosen canonical model and is thus well-defined.

Example 22 Consider again the set of guarded normal TGDs £ and the two normal BCQs Q1 and Q2 of Example 19. Let the database D be given as in Example 21. Then, by the canonical model S2 shown in Example 21, Q1 and Q2 are answered positively and negatively, respectively.

10.3 Query Answering

As we have seen in the previous section, a canonical model of a database and a stratified set of guarded normal TGDs can be determined via iterative chases, where every chase may be infinite. We next show that for answering safe normal BCQs, it is sufficient to consider only finite parts of these chases. Based on this result, we then show that answering safe normal BCQs in guarded (resp., linear)

[redacted]

[redacted]

[redacted]

[redacted]

Datalog± with stratified negation is data-tractable (resp., FO-rewritable).	
We first give some preliminary definitions as follows. Given a set of atoms S, a database D, and a set of guarded normal TGDs £, we denote by chaseS(D, £) a slightly modified oblivious chase where the TGD chase rule is applicable on a guarded normal TGD a iff the homomorphism h maps every atom in body-(a) to an atom not from S, and in that case, the TGD chase rule is applied on the TGD obtained from a by removing the negative body of a. Then, g-cha,sel,S(D, £) denotes the set of all atoms of depth at most l in the guarded chase forest.	[redacted]
The next result shows that safe normal BCQs Q can be evaluated on finite parts of iterative guarded chase forests of depths depending only on Q and R. Its proof is similar to the proof of Lemma 4. The main difference is that the atoms of Q may now belong to different levels of a stratification, and one also has to check that the negative atoms do not match with any atom in a canonical model.	[redacted]
Theorem 32 Let R be a relational schema, D be a database for R, £ be a stratified set of guarded normal TGDs on R, and Q be a safe normal BCQ over R. Then, there exists some l ^ 0, which depends only on Q and R, such that D U £ l=strat Q iff Q evaluates to true on Sk, where the sets Si,, i € {0,... ,k}, are defined as follows:	[redacted]
(i) S0 = g-chase1 (D, £0); (ii) if i > 0, then S, = g-chasei,Si-1 (S,—i, £,).	[redacted]
The following result shows that	[redacted]

answering safe normal BCQs in guarded Datalog± with stratified nega¬tion is data-tractable. Like in the negation-free case, not only homomorphic images of the query atoms are contained in finite portions of iterative guarded chase forests, but also the whole derivations of these images. That is, the theorem is proved similarly as Theorems 5 and 6; the main difference is that the finite portion of the guarded chase forest is now computed for each level of a stratification, and that we now also have to check that the negative atoms cannot be homomorphically mapped to a canonical model.	[redacted]
Theorem 33 Let R be a relational schema, D be a database for R, £ be a stratified set of guarded normal TGDs on R, and Q be a safe normal BCQ over R. Then, D U £ |=Q is decidable in polynomial time in the data complexity.	[redacted]
The next result shows that answering safe normal BCQs in linear Datalog± with stratified negation is FO-rewritable. Its proof extends the line of argumentation in Theorem 9 and Corollary 10 by stratified negation and safe normal BCQs.	[redacted]
Theorem 34 Let R be a relational schema, £ be a stratified set of linear normal TGDs on R, and Q be a safe normal BCQ over R. Then, Q is FO-rewritable.	[redacted]
10.4 Perfect Model Semantics and Independence from Stratification	[redacted]
We now introduce the perfect model semantics of guarded Datalog± with stratified negation, and prove that it coincides with the canonical model semantics (thus also being	[redacted]

independent of a concrete stratification), and that it is an isomorphic image of the perfect model semantics of the corresponding normal logic program with function symbols [74]. This gives strong evidence that the canonical model semantics of guarded Datalog± is quite natural.

[redacted]

The perfect model semantics of databases D and a stratified set of guarded normal TGDs £ is defined via a preference relation ^ on the models of D and £ that are isomorphic images of models of D and £f. We first define the strict and reflexive relations -< and ^ on ground atoms (having terms with function symbols as arguments). Given a set of guarded normal TGDs £, the relations -< and ^ on the set of all ground atoms are the smallest relations that satisfy (i) to (iv):

[redacted]

(i) ^(head (ct)) ^ ^(a) for every ct € ground (£f) and every a € body+(ct),

[redacted]

(ii) ^(head (ct)) -< ^(a) for every ct € ground (£f) and every a € body—(ct),

[redacted]

(iii) -< and ^ are transitively closed, and

[redacted]

(iv) -< is a subset of ^.

We are now ready to define the preference relation ^ on isomorphic images of models of D and £f, as well as the perfect model semantics of D and £ as a collection of isomorphic such images that are preferred to all others.

[redacted]

Definition 6 Let R be a relational schema, D be a database for R, and £ be a set of guarded normal TGDs on R. For isomorphic images M, N C HB£ of two models Mf and Nf of D

[redacted]

and £f, respectively, we say that M is preferable to N, denoted M ^ N, iff (i) M is not isomorphic to N and (ii) for every a € Mf — Nf, there exists some b € Nf — Mf such that a -< b (which is also denoted Mf ^ Nf).	[redacted]
Given an isomorphic image M C HB^ of a model of D and £f, we say that M is a perfect model of D and £ iff M ^ N for all isomorphic images N C HB£ of models of D and £f such that N is not isomorphic to M.	[redacted]
The following lemma shows that the preference relation ^ is well-defined.	[redacted]
Lemma 35 Let R be a relational schema, D be a database for R, and £ be a set of guarded normal TGDs on R. Let Mf, Mf, Nf, and Nf be models of D and £f, let M C HBs (resp., N C HBs) be an isomorphic image of Mf and Mf (resp., Nf and Nf). Then, Mf ^ Nf iff Mf ^ Nf.	[redacted]
The following result shows that in the negation-free case, perfect models of D and £ are isomorphic to the least model of D and £f. Hence, by Theorem 29, they are also isomorphic to the oblivious chase of D and £.	[redacted]
Proposition 36 Let R be a relational schema, D be a database for R, and £ be a set of guarded TGDs on R. Then, M is a perfect model of D and £ iff M is an isomorphic image of the least model of D and £f.	[redacted]
The next result shows how perfect models of D and £ can be iteratively constructed. Here, given a stratification ^: R ^ {0,1,... ,k} of £, we define HB i (resp., HB *) as the set of all a € HB s with ^(pred(a)) = i (resp., ^(pred(a)) ^ i).	[redacted]
Proposition 37 Let R be a relational schema, D be a database for R, and £	[redacted]

be a set of guarded nor¬mal TGDs on R with stratification ^: R ^ {0,1,... ,k}. Let S C HB * and S' C HB *+1 such that S' n HB* = S. Then, for all i € {0,1,..., k—1}, S' is a perfect model of D*+1 and £*+1 iff (i) S is a perfect model of D* and £*, and S is an isomorphic image of a model Sf of D* and (£*)f, and (ii) S' is an isomorphic image of the least model of Sf U Di+1 and (£f+1)sf.	[redacted]
Observe that the perfect model of the normal logic program D U £f is given by Mk, where M0 is the least model of D0 U £0, and every M i+1, i € {0,1,... ,k — 1}, is the least model of Mi U Di+1 U (£f+1)M". Hence, as an immediate corollary of Propositions 36 and 37, we obtain that the perfect model of D and £ is an isomorphic image of the perfect model of D and £f.	[redacted]
Corollary 38 Let R be a relational schema, D be a database for R, and £ be a stratified set of guarded normal TGDs on R. Then, M is a perfect model of D and £ iff M is an isomorphic image of the perfect model of D and £f.	[redacted]
The following theorem shows that the perfect model semantics coincides with the canonical model semantics. It is proved using Theorem 29 and Propositions 36 and 37. Since perfect models are independent of a concrete stratification, the theorem also implies that the same holds for the canonical model semantics.	[redacted]
Theorem 39 Let R be a relational schema, D be a database for R, and £ be a stratified set of guarded normal TGDs on R. Then, M is a canonical model of D and £ iff M is a perfect	[redacted]

model of D and £.

11 Related Work

In this section, we give a brief overview of several related approaches in the literature. The main motivation behind our research is the Semantic Web. We recall that the vision of the Semantic Web has led in the recent years to a new way of conceiving information systems, deeply integrated into the Web and its semantics, where Web information is annotated, so as to be machine-readable; in this way, such information can be integrated and especially queried in information systems, and not merely searched by keywords. This re¬quires a precise sharing of terms by means of an ontology, so that the semantics of terms across different sources is clear. Moreover, by using ontologies, it is possible to perform automated reasoning tasks in order to infer new knowledge from the raw information residing on the Web. Underneath the ontology, a data layer represents the raw data present on the Web, in an inherently heterogeneous way. The World Wide Web Consortium (W3C) defines several standards, including the Resource Description Framework (RDF) for the data layer, the Web Ontology Language (OWL) (based on description logics (DLs)) for the ontology layer, and the currently being standardized Rule Interchange Format (RIF) for the rule layer. As for the latter, rather than providing a common semantics, RIF aims at offering a common exchange format for rules, given that numerous languages already exist.

We start by discussing the features of

[redacted]

[redacted]

[redacted]

DLs employed in Semantic Web reasoning. We then review a few works in database theory that are deeply related to Semantic Web reasoning, highlighting the role of the chase. Datalog is a well-known language for knowledge bases, and we briefly describe some of its extensions beyond the languages presented in this paper, which can also play a prominent role in ontolog¬ical querying. We also discuss different rewriting techniques that have been the subject of several relevant research activities on ontology querying. We then describe different approaches to integrate rules and on¬tologies. After reviewing some ontology reasoning systems, we close by discussing the issue of whether to consider infinite models for the theories constituted by the ontologies.

[redacted]

11.1 Description Logics

In the Semantic Web, the ontology layer is highly important, and has led to a vast corpus of literature. DLs have been playing a central role in ontology reasoning; they are decidable fragments of first-order logic, based on concepts (classes of objects) and roles (binary relations on concepts); several variants of them have been thoroughly investigated, and a central issue is the trade-off between expressive power and computational complexity of the reasoning services. In DL reasoning, a knowledge base usually consists of a TBox (terminological component, i.e., ontology statements on concepts and roles) and an ABox (assertional component, i.e., ontology statements on instances of

[redacted]

concepts and roles); the latter corresponds to a data set. The description logic SROIQ [56] is one of the most expressive DLs, which is underlying OWL 2 [103] , a new version of OWL [102]. Reasoning in SROIQ is computationally expensive, and several more tractable languages have been proposed in the Semantic Web community. Among such languages, we now discuss the DL-Lite family [34, 89], EL++ [11], and DLP [54], which are underlying the OWL 2 profiles QL, EL, and RL [104], respectively, as well as ELP [69], SROEL(x) [65], and SROELV3(n, x) [66].	[redacted]
The DL-Lite family of description logics [34, 89] focuses on conjunctive query answering under a database and a set of axioms that constitute the ontology; query answering is in AC0 in the data com¬plexity, due to FO-rewritability of all languages in the DL-Lite family (note that query answering in the extended DL-Lite family introduced in [8, 9] may also be more complex (P and co-NP)). The description logic DL-LiteR of the DL-Lite family provides the logical underpinning for the OWL 2 QL profile [104]. Note here that the unique name assumption can be given up in DL-LiteR and OWL 2 QL, as it has no impact on the semantic consequences of a DL-LiteR and an OWL 2 QL ontology. In addition to being strictly more expressive than DL-Lite, linear Datalog± (with negative constraints and non-conflicting keys) is also strictly more expressive than OWL 2 QL.	[redacted]

The description logic EL++ [11] is an extension of EL [10, 11] by the bottom element ±, nomi- nals, concrete domains, and role inclusions (between concatenations of abstract roles and atomic abstract roles); reasoning in EL++ is PTIME-complete, while conjunctive query answering in EL++ is undecid- able. Since guarded Datalog± can express EL, guarded Datalog± (with negative constraints) can also express all the above extensions of EL except for role inclusions with a concatenation of abstract roles on the left-hand side, which require non-guarded rules. The OWL 2 EL profile [104] is based on EL++; reasoning and conjunctive query answering in OWL 2 EL are both PTIME-complete in the data complex¬ity. As OWL 2 EL allows for stating the transitivity of atomic roles, it is also not expressible in guarded Datalog±. Note that both EL++ and OWL 2 EL, differently from guarded Datalog±, also do not make the unique name assumption; however, guarded Datalog± (with negative constraints and non-conflicting keys) can easily be extended (without increase of complexity) in this direction by abstracting from constants to equivalence classes of constants denoting the same objects (this then requires to compute the reflexive-symmetric-transitive closure of the "same as" relation, which can be done in polynomial time). Interestingly, differently from OWL 2 QL, OWL 2 EL also allows to express TGDs of the form p(X) ^ q(X, X) (via ax¬ioms "SubClassOf (:p, ObjectHasSelf (:q))").	[redacted] [redacted] [redacted]

DLP [54] is a Horn fragment of OWL, i.e., a set of existential-free rules and negative constraints, with¬out unique name assumption. Since these rules may be non-guarded, DLP is not expressible in guarded Datalog± (with negative constraints and non-conflicting keys). The OWL 2 RL profile [104] is an (existen¬tial-free) extension of DLP, which aims at offering tractable reasoning services while keeping a good ex¬pressive power, enough to enhance RDF Schema with some extra expressiveness from OWL 2. Compared to DLP, OWL 2 RL can in particular additionally encode role transitivity, which is also not expressible in guarded Datalog±. Conversely, due to the missing existential quantifiers in rule heads, guarded Datalog± is clearly neither expressible in DLP nor in OWL 2 RL.	[redacted]
The rule-based tractable language ELP [69] generalizes both EL++ and DLP. In particular, it extends EL++ with local reflexivity, concept products, universal roles, conjunctions of simple roles, and limited range restrictions. Here, concept products are another source of non-guardedness, in addition to the sources already inherited from EL++ and DLP. Thus, ELP is not expressible in guarded Datalog±.	[redacted]
A closely related extension of EL++ is the DL SROEL(x) [65], which provides efficient rule-based inferencing for OWL 2 EL, and which is in turn extended by the DL SROELV3(n, x) [66]. The latter introduces so-called nominal schemas, which allow for variable	[redacted]

nominals, which are expressions that may appear in more than one conjunct in a concept expression, and such that all occurrences of the same variable nominal bind to the same individual. This way, to represent a SROELV3(n, x) assertion with a TGD, we need in general arbitrary joins, which are also beyond the expressiveness of guarded Datalog±. Observe that, conversely, guarded Datalog± allows for predicate symbols of arbitrary arity n ^ 0, while SROELV3(n, x) in its original version in [66] (and also SROEL(x), ELP, OWL 2 EL, and EL++) only allows for predicate symbols of arity n ^ 2 (but can be extended to predicate symbols of arbitrary arity n ^ 0). Another important difference between SROELV3(n, x) and guarded Datalog± is that SROELV3(n, x) covers (restricted versions) of the profiles OWL 2 EL and OWL 2 RL, but does not cover the profile OWL 2 QL, while guarded Datalog± strictly covers OWL 2 QL, but does not cover OWL 2 EL and OWL 2 RL.

11.2 Database Schemata, Ontologies, and Chase

Classical database constraints, as well as more involved ones, can be employed in modeling complex schemata and ontologies. Interestingly, the well-known inclusion dependencies [1], common constraints in relational databases, are quite useful in expressing ontologies; linear TGDs are in fact an extension of inclusion dependencies. In [25, 27], inclusion dependencies are employed together with key dependencies to represent

[redacted]

[redacted]

an extension of entity-relationship schemata; FO-rewritable subclasses of such schemata are defined by means of graph-based conditions. The chase [79, 58, 30] of a database against a set of inclusion dependencies is crucial in ontological query answering; it has been extended to TGDs in [49, 41]. In most practical cases in ontological reasoning, the chase does not terminate; the first work to tackle the problem of a non-terminating chase was [58]. In data exchange, the chase is necessarily finite; weakly-acyclic sets of TGDs are the main class of sets of TGDs that guarantees chase termination [49], later extended by [81,41]. However, this is not appropriate for ontological databases.

11.3 Datalog Extensions

Datalog [1] is a powerful language, but it has some inherent limitations in modeling ontologies, as clearly discussed in [87]. To overcome such limitations, existential quantification was introduced in Datalog (Horn) rules in the form of value invention [80, 21]. Datalog rules with value invention are in fact TGDs. Guarded TGDs in guarded Datalog± are extended by weakly-guarded TGDs in weakly-guarded Datalog± [23]. The restrictions on the bodies of guarded and weakly-guarded TGDs, respectively, however, cannot express, e.g., the concept product [63, 64, 68]; they also cannot capture assertions having compositions of roles in their body, which are inherently non-guarded. A recent paradigm called stickiness, on which sticky Datalog± and its variants are

[redacted]

[redacted]

based, allows for such rules. Sticky sets of TGDs [26, 28] are defined via a condition based on variable marking. Like DL-Lite, some of the variants of sticky Datalog± are FO-rewritable, and therefore tractable. Sticky Datalog± also properly extends DL-Lite.	[redacted]
Recent works focus on general semantic characterizations of sets of TGDs for ensuring decidability of query answering. One such characterization is the notion of finite unification set (FUS), which is strictly connected to that of rewriting. Given a set of TGDs £ and a BCQ Q, a backward chaining mechanism is a procedure that constructs a rewriting Qs of Q relative to £, also called £-rewriting of Q, such that for every database D, D U £ |= Q iff D |= Qs. The key operation in backward chaining is the unification between the set of atoms in the body of Q and the head of some TGD in £; for the precise definitions, see [13, 84]. Finite unification sets (FUSs) ensure that the constructed rewriting is finite. Thus, query answering under FUSs is decidable, as we just have to build the (finite) rewriting, and then evaluate it over the given database. Interestingly, linearity and stickiness are sufficient syntactic properties which ensure that the TGDs are FUSs [26].	[redacted]
Another such characterization is the notion of bounded treewidth set (BTS). A set of TGDs £ is called bounded treewidth set (BTS) iff for every database D, the chase graph of chase(D, £) has bounded treewidth. Intuitively, this means that the chase	[redacted]

graph is a “tree-like” graph. Decidability of query answer¬ing under BTSs was established in [23]. A FUS is not necessarily a BTS (e.g., sticky TGDs). Every set of linear, guarded, and weakly guarded TGDs is a BTS [23]. Two other classes of sets of TGDs that are BTSs are sets of frontier-guarded and of weakly-frontier-guarded TGDs, where the latter generalize both sets of guarded and of frontier-guarded TGDs [13, 84]. Intuitively, the frontier of a TGD is the set of variables that are shared between the body and the head, and a TGD is frontier-guarded iff an atom in its body guards the frontier. Frontier-guardedness also allows tractable query answering in the data complexity [14].	[redacted]
Under certain conditions [13], a FUS can be combined with a BTS while retaining decidability of query answering. A class of sets of TGDs that are neither FUSs nor BTSs, but still ensure decidable (and even data-tractable) query answering, are sets of weakly-sticky TGDs [28], which generalize sets of sticky TGDS.	[redacted]
Less closely related, [47] presents a class of logic programs with function symbols, disjunction, con¬straints, and negation under the answer set semantics; consistency checking and brave/cautious reasoning are shown to be EXPTlME-complete in this setting, and some lower-complexity fragments are presented.	[redacted]
11.4 Query Rewriting Ontological queries are often answered through some form of query reformulation (or rewriting) that (par¬tially) embeds the constraint of the TBox into the query	[redacted]

itself to avoid the explicit generation of the universal model. The rewriting algorithm for the FO-rewritable languages of the DL-Lite family is based on back¬ward resolution, as others developed for dealing with entity-relationship schemata [22], inclusion depen¬dencies [31], and linear TGDs [51]. Such rewriting algorithms produce a first-order rewriting in form of a union of conjunctive queries (UCQs). The work in [88] instead goes beyond FO-rewritability by proposing a Datalog rewriting algorithm for the expressive DL ELHIO-, which comprises a limited form of concept and role negation, role inclusion, inverse roles, and nominals, i.e., concepts that are interpreted as single¬tons; conjunctive query answering in ELHIO- is PTIME-complete in the data complexity. The proposed algorithm is based on resolution and produces an "optimal" rewriting relative to the language adopted to define the TBox constraints. In particular, in the case of DL-Lite, it produces a UCQs FO-rewriting instead of a Datalog rewriting.

Other rewriting techniques for PTIME-complete languages (in the data complexity) have been proposed for EL [95, 62, 67, 78]. Another approach to rewriting is a combination of rewriting according to the ontology and to the data; this was proposed in [60,61] for DL-Lite. As already noticed in [88,60], rewritings in the form of a UCQs may have a size that is exponential in the size of the TBox and of the input query. To overcome this problem,

[redacted]

[redacted]

several techniques have recently been proposed. The work in [60, 61] adopts a combined approach that first extends the input database with additional facts that “witness” the existential constants of the TBox and then rewrites the input query into a first-order query that retrieves only the sound answers. This technique is best suited in those cases where the rewriting according to the TBox alone is very large; however, it intrinsically depends on the data and is therefore not a purely intensional first-order rewriting. In [96], the problem of the exponential size of the UCQs rewriting techniques for DL-Lite is addressed by targeting a non-recursive Datalog program as a form of the rewriting. The size of the rewriting remains polynomial in the size of the TBox, but still exponential in the size of the input query. In [85], the ideas at the basis of [88] are refined to obtain a predicate-bounded Datalog rewriting for ontologies expressed with linear TGDs. Also in this case, the worst-case size of the rewriting is exponential in the size of the input query, but polynomial in the size of the TBox. Recently, [53] proposed a rewriting technique that produces a non-recursive Datalog (and therefore first-order) rewriting that is of polynomial size relative to the size of the input query and the TBox. This algorithm applies to classes of TGDs that enjoy the polynomial-witness property [53] among which there are linear TGDs and, therefore, DL-Lite. An up-to-date survey of rewriting approaches for ontological query answering can be found in [52].

[redacted]

11.5 Integration of Rules and Ontologies

[redacted]

The integration of rules with ontologies has recently raised significant interest in the research community, as it allows for combining the high expressive power of rule languages with the interoperability provided by ontologies. This is different from the approach of this paper, where we concentrate on the ontology alone (expressed as rules), aiming at attaining the highest expressive power with the least computational cost. Essentially, we can classify the approaches to this integration into two categories: loose coupling (or strict semantic separation) and tight coupling (or strict semantic integration).

[redacted]

Loose coupling. In loose coupling, the rules layer consists of a (usually) nonmonotonic language, while the ontology layer is expressed in OWL/RDF flavor. The two layers do not have particular restrictions, as their interaction is forced to happen through a "safe interface": rule bodies contain calls to DL predicates, allowing for a mix of closed- and open-world semantics. An example of this approach are dl-programs, together with several extensions [45, 46, 43, 44, 76, 77, 100], including probabilistic dl-programs, fuzzy dl-programs, and HEX-programs. More precisely, probabilistic and fuzzy dl-programs [76] extend dl-pro- grams by probabilistic uncertainty and fuzzy vagueness, respectively, while HEX-programs [45, 46] extend the framework of dl-programs so that

[redacted]

they can integrate several different external sources of knowledge, possibly under different semantics. A framework for aligning ontologies is added on top of dl-programs in [100]. The work in [105] extends dl-programs to handle priorities. Defeasible reasoning is combined with DLs in [6]; in this work, like in the above cited ones, the DL layer serves merely as an input for the default reasoning system; a similar approach is followed in the TRIPLE rules engine [98].

Tight coupling. In tight coupling, the existing semantics of rule languages is adapted directly in the ontol¬ogy layer. The above cited DLP [54] is an example of this, as well as the undecidable SWRL [57]; a mutual reduction between inference in a fragment of the DL SHOIQ and a subset of Horn programs is shown in [54] . Between DLP and SWRL, several other works extend the expressiveness while retaining decidability, dealing with this trade-off in different ways. Among hybrid approaches, in which DL knowledge bases act as input sources, we find the works [42, 72, 91, 92]. The paper [42] combines plain Datalog (without negation or disjunction) with the DL ALC, obtaining a language called AL-log. In AL-log, concepts in an ALC knowledge base (the structural component) enforce constraints in rule bodies of a Datalog program (the relational component). Levy and Rousset in [72] present the carin framework, which combines the DL ALCNR with logic programs in a similar fashion, allowing also roles to

[redacted]

[redacted]

enforce constraints on rules (unlike [42] which allows only concepts to impose constraints). Such interaction leads to undecidability eas¬ily, but in [72] two decidable fragments are singled out. Another work along the same lines is [83]. Rosati's r-hybrid knowledge bases [91, 92] combine disjunctive Datalog (with classical and default negation) with ALC based on a generalized answer set semantics; besides the satisfiability problem, also that of answering ground atomic queries is discussed. This formalism is the basis for a later one, building upon it, called DL + log [93]. Another approach is found in [82], in the framework of hybrid MKNF knowledge bases, based on the first-order variant of Lifschitz's logic MKNF [73]. Other recent approaches to combine rules and ontologies through uniform first-order nonmonotonic formalisms are found in [75, 20].

11.6 Systems

Several systems perform reasoning services on ontologies in various flavors. The CEL system [12] is based on EL+, i.e., EL extended by role inclusions; CEL can perform subsumption in polynomial time, thus aiming at tractable reasoning on large knowledge bases. Snomed [37] is also based on EL, restricted to having acyclic TBoxes only. Snorocket [70] is based on EL++, and achieves good scalability without the restrictions of SNOMED. The research on DL-Lite has also given raise to systems, in particular, QuOnto [3], Mastro [34], and Requiem [88]; they rewrite queries into SQL according to a DL-Lite

[redacted]

[redacted]

TBox, thus taking advantage of the optimizations of an underlying relational DBMS. In particular, Requiem accepts as input more expressive languages such as ELHIO-. The linear, guarded, and sticky-join languages of Datalog± are supported by the Nyaya knowledge management system [40]. Apart from these research prototypes, there are also commercial systems that deal with more expressive languages. Owlim [18], IBM IODT [106], and Oracle 11g [101] allow to store, reason over, and query large OWL-DL ontologies. The Pellet and Racer-pro reasoners [86, 55] also partially support conjunctive query answering for OWL-DL ontologies.

11.7 Finite Controllability

Finally, we have considered in this paper entailment under arbitrary (finite or infinite) models; when this coincides with entailment under finite models only, it is said that finite controllability [58, 94, 15] holds.

12 Conclusion

We have introduced a family of expressive extensions of Datalog, called Datalog±, as a new paradigm for query answering and reasoning over ontologies. The Datalog± family admits existentially quantified variables in rule heads, and has suitable restrictions to ensure highly efficient ontology querying. These languages are rather attractive, as they are simple, easy to understand and analyze, decidable, and they have good complexity properties. Furthermore, for ontological query answering and reasoning, they turn

[redacted]

[redacted]

[redacted]

out to be extremely versatile and expressive: in fact, guarded Datalog± can express the tractable description logic EL, and languages as simple as linear Datalog± with negative constraints and NC keys (both simple first-order features) can express the whole DL-Lite family of tractable description logics (including their generalizations with n-ary relations), which are the most popular tractable ontology languages in the context of the Semantic Web and databases. We have also shown how nonmonotonic stratified negation (a desirable expressive feature that DLs are currently lacking) can be added to Datalog±, while keeping ontology querying and reasoning tractable.	[redacted]
Datalog± is the first approach to a generalization of database rules and dependencies so that they can express ontological axioms, and it is thus a first step towards closing the gap between the Semantic Web and databases. Datalog± paves the way for applying decades of research in databases, e.g., on data integration and data exchange, to the context of the Semantic Web, where there is recently a strong interest on highly scalable formalisms for the Web of Data.	[redacted]
The Datalog± family is of interest in its own right; it is still a young research topic, and there are many challenging research problems to be tackled. One interesting topic is to explore how Datalog± can be made even more expressive. For example, many DLs allow for restricted forms of transitive closure or constraints. Transitive closure is easily expressible in Datalog, but only	[redacted]

through non-guarded rules, whose addition to decidable sets of rules may easily lead to undecidability. Hence, it would be interesting to study under which conditions, closure can be safely added to various versions of Datalog±. Moreover, for those logics where query answering is FO-rewritable, the resulting FO-query is usually very large. A topic for future work is to study from a theoretical and a practical point of view how such FO-rewritings can be optimized. Furthermore, it would also be interesting to explore how more general forms of nonmonotonic negation, such as negation under the well-founded and under the answer set semantics, can be added to Datalog±, which could, e.g., also be applied when combining/merging two ontologies, where the underlying stratifications cannot always be maintained. Finally, SPARQL [5] has the same expressive power as non¬recursive Datalog with negation; capturing the integration of SPARQL and DL-Lite seems feasible with extensions of Datalog±, which will be another subject of future investigation.

Appendix A: Proofs for Section 3

Lemma 1 Let R be a relational schema, D be a database for R, and £ be a set of guarded TGDs on R. Let S be a finite subset of A U AN, and let ai and a2 be atoms from chase (D, £) such that (ai, type (ai)) and (a2, type(a2)) are S-isomorphic. Then, the subtree of ai is S-isomorphic to the subtree of a2.

Proof. We give a proof by induction on the number of applications of the

[redacted]

[redacted]

[redacted]

TGD chase rule to generate subtree(a1) and subtree(a2).	[redacted]
Basis: We apply the TGD chase rule to generate a child of the nodes labeled with a1 and a2 in the subtrees. The side atoms in such applications are contained in type(a1) and type(a2), respectively. Suppose that we are adding a node labeled with an atom bi as a child of the node labeled with a1, applying a TGD a € £, and using as side atoms S1 C type(a1). Then, there is another set S2 C type(a2) that is S-isomorphic to S1. Hence, we can apply a to a2 using S2 as side atoms, and we obtain a node labeled with an atom b2 as a child of the node labeled with a2, which is S-isomorphic to b1. Thus, we can extend the S-isomorphism between type(a1) and type(a2) to an S-isomorphism between type(a1) U {b1} and type(a2) U {b2} by assigning to every fresh null in b1 the corresponding fresh null in b2.	[redacted]
Induction: By the induction hypothesis, type(a1) U P1 and type (a2) U P2 are S-isomorphic, where every Pi, i € {1,2}, is the set of atoms introduced in subtree(ai) during the first k applications of the TGD chase rule. The proof is now analogous to the one of the basis, replacing every type(ai), i € {1, 2}, by type(ai) U Pi, and considering the (k + 1)-th application of the TGD chase rule. □	[redacted]
Lemma 2 Let R be a relational schema, D be a database for R, and £ be a set of guarded TGDs on R. Let w be the maximal arity of a predicate in R, 5 = |R| ■ (2w)w ■ 2]]'R]]'(2w^w, and a € chase (D, £).	[redacted]

Let P be a set of pairs (b, S), each consisting of an atom b and its type S of atoms c with arguments from a and new nulls. If |P | > 5, then P contains at least two dom (a)-isomorphic pairs.	[redacted]
Proof. As for the atoms b, at most w arguments from a and at most w nulls in the two extreme cases may be used as arguments in b. We thus obtain 2w possible symbols, which can be placed into at most w argument positions of |R| predicates. Hence, the number of all non- dom (a)-isomorphic atoms b is given by |R| ■ (2w)w. As for the types S, we thus obtain 2]]'R]]'(2w')W as the number of all subsets of this set of atoms. In summary, the number of all pairs as stated in the theorem is bounded by 5 = |R| ■ (2w)w ■ 2]]'R']]'(2w')W. □	[redacted]
For the proof of Lemma 3, we need some preliminary definitions and results as follows. Each proof n of a ground atom a from a database D and a set of guarded TGDs £ gives rise to a guarded proof forest GPF n, which is the smallest subforest of the guarded chase forest for D and £ containing (i) a vertex labeled with b for each atom b of D that is also a vertex of n, and (ii) for all sets of vertices {b1,..., br, b} of n such that there is a TGD a € £ that applied to b1,..., br produces b, where bi unifies with the guard of a, a vertex labeled with bi, a vertex labeled with b, and an arc between the former and the latter. The n-depth of an atom b in n, denoted n-depth(b), is its smallest depth in GPFn. Note that for each atom b occurring in n, it holds that depth(b) ^ n-depth(b).	[redacted]

For a relational schema R and set of terms (constants or variables) T, the atom base of T, denoted ABR(T), is the set of all atoms that can be built from predicate symbols in R and terms in T. For a ground atom a over R, the Herbrand base of a, denoted HBR(a), is defined by HBR(a) = ABR(dom(a)).	[redacted]
Let R be a relational schema, £ be a set of TGDs on R, d be a ground atom over R, and S be a set of ground atoms over R. Then, a (d, S, £)-proof of a ground atom e is defined as a proof n of e from {d} U S and £, where elements of S are used as side atoms only, and whose associated guarded proof forest GPFn is a thus a single tree whose root is labeled d.	[redacted]
Lemma 40 Let R be a relational schema, and £ be a set of guarded TGDs on R. Let d and e be ground atoms over R such that e € HBR(d), and S be a set of ground atoms over R such that S C HBR(d). Let P(R, £, S, d, e) be defined by:	[redacted]
• ft (R, £, S, d, e) = 0, if there is no (d, S, £) -proof of e;	[redacted]
• ft(R, £, S, d, e) = k, where k is the smallest integer i such that there is a (d, S, £)-proof n whose atoms are all in g-chase*({d} U S, £), otherwise.	[redacted]
Then, there is an upper bound y on ft(R, £, S, d, e) that depends only on R, where 7 is double-exponentially (resp., single-exponentially) bounded in R in the general case (resp., in the case of a fixed arity).	[redacted]
Proof. Let w be the maximum predicate arity of R. Let C = {ci,..., cw } be a set of arbitrary distinct data constants. Note that the choice of C	[redacted]

is completely irrelevant, and we could as well use symbolic dummy constants. A trivial upper bound ft'(R, £) for ft(R, £, S, d, e) that depends only on R and £ is defined by:

ft'(R, £) = max ft(R, £,S', d', e').
d'eHB (C)
S'CHB (d') e'eHB (d')

Observe now that, unlike for general TGDs, for each given schema R, there is (up to isomorphism) only a finite set r(R) of guarded TGDs over R, and this set depends solely on R. In fact, each guarded TGD has a guard, and there is a clearly determined finite choice of guard predicates in R. After having chosen a guard predicate P of arity r, there are no more than rr ^ possibilities of populating its arguments with variables from X = {xi, x2,..., xr}. Let V C X be the set of variables used in the guard. Then, the set of side atoms can only consist of a subset of ABR(V), which is clearly a finite set. In a similar way, the set of possible choices for the head of a guarded TGD is finite and determined by the guard. In summary, r(R) is (up to isomorphic variable renaming) unique, finite, and determined by R. It follows that there are (up to isomorphism) only finitely many sets of guarded TGDs £ over R, and their totality is determined by R. In fact, these possible sets of guarded TGDs are the subsets of r(R). Note that the number of all (non-isomorphic) guarded TGDs formed according to R is at most double-exponential in the size of R. Each £' C r(R) is thus at most double-exponential in the

[redacted]

[redacted]

size of R. The above allows us to obtain an upper bound y(R) for ft(R, £, S, d, e) that depends only on R:

Y(R) = max ft'(R, £') . s'cr(R)

The above line of argumentation does not give us a concrete bound of Y(R) in terms of the size of R. It does not even show that Y(R) is computable from R. To understand how Y(R) depends on R, we may, however, analyze a suitably restricted version of the (alternating) Acheck algorithm presented in the extended version of [23]. The analysis shows that Y(R) is double-exponentially bounded in the size of R in the general case and exponentially in case of bounded arities. A rough sketch of such an analysis goes as follows. We may concentrate on the setting where a database atom d is given and a set of ground atoms S has already been shown to be derivable at the required depth. The Acheck algorithm, when adapted to our setting, generates on input {d} U S and £ a guarded chase tree rooted in d, and checks whether a belongs to it. Each (macro-)configuration of Acheck actually corresponds to a vertex of this guarded chase tree. Given that the size of each configuration of Acheck is at most exponential in the size of R, in case the atom a is at all contained in the guarded chase tree, it must be derivable within a double-exponential number of chase steps. In the case of bounded arities, the Acheck configurations are each of size polynomial in the size of R, and thus there exists a derivation of a in exponentially many chase steps. □

Lemma 3 Let R be a relational

[redacted]

[redacted]

schema, D be a database for R, and £ be a set of guarded TGDs on R. Then, there is a constant y, depending only on R, such that for each ground atom a € chase (D, £), there is a proof of a from D and £, whose atoms all belong to g-chase1 (D, £).	[redacted]
Figure 5: Construction in the proof of Lemma 3.	[redacted]
Proof. Assume that a is a ground atom such that a € chase (D, T). We use induction on the derivation level 1(a) of a in chase(D, T) to show that a has a proof from D and £ that lies entirely in g-chaseY(D, T), where 7 = 7(R) as defined in the proof of Lemma 40.	[redacted]
Basis: For 1(a) = 0, it holds that a belongs to D, and thus there is a proof of a from D and T of depth 0.	[redacted]
Induction: Assume now that for all ground atoms b of derivation level 1(b) < n, there is a proof of b lying entirely in g-chase1 (D, T), and assume that 1(a) = n. Note that the guarded chase forest F for a may contain several vertices labeled a, corresponding to different possible derivations of a. Among those, we choose one vertex v that was generated according to a derivation of a of minimum level. Let Ta be the tree in the guarded chase forest containing vertex v. Then, Ta is rooted in some vertex vo labeled with some ground atom d € D. Note that, due to guardedness, it must hold that a € type(d). In fact, no constant that was not already present in the root can enter Ta. Consider now a minimal set S of all side atoms a' of	[redacted]

atoms in Ta that contribute to the generation of vertex v, that are of derivation level l(a') <l(a), and that are not generated within Ta at derivation level l(a'). These are the side atoms used to generate node v labeled a in Ta that come from trees different from Ta in the forest F. Due to the guardedness of T, these side atoms are all ground and belong to type(d). By the induction hypothesis, having a lower derivation level than a, each atom a' € S has a proof na/ that lies entirely in g-chaseY(D, T). Observe that v in Ta is generated by a chase derivation from {d} U S and T, whose guarded chase forest is rooted in d. Moreover, a € type(d) C HBR(d) and S C type(d) C HBR(d). Hence, the precondition of Lemma 40 applies, and thus there exists a (d, S, T)-proof of a of depth 7 = 7(R). This proof, jointly with all proofs of atoms a' € S, constitutes a complete proof of a from D and T that lies entirely in g-chaseY(D, T). □	[redacted]
Lemma 4 Let R be a relational schema, D be a database for R, T be a set of guarded TGDs on R, and Q be a BCQ over R. If there exists a homomorphism ^ that maps Q into chase (D, T), then there exists a homomorphism X that maps Q into g-chasek(D, T), where k depends only on Q and R.	[redacted]
Proof. Let k = n ■ 5, where n = |Q|, 5 = |R| ■ (2w)w ■ 2\n\'(2'ww, and w is the maximal arity of a predicate in R. Suppose there exists a homomorphism that maps Q into chase (D, T). Let ^ be a homomorphism of this kind such that depth(^) = Y1 q&Q depth(y(q)) is minimal. We now show that y(Q) is	[redacted]

contained in g-chasek(D, T). Towards a contradiction, suppose the contrary. Consider the tree consisting of all nodes labeled with atoms in y(Q) and their ancestors in the guarded chase forest for T and D. Since y(Q) is not contained in g-chasek(D, T), this tree must contain a path P of length greater than 5 of which the labels (= atoms) of all inner nodes (i.e., without start and end node) do not belong to y(Q) and have no branches (i.e., have exactly one outgoing edge). Let the atom a be the label of the start node of P. By Lemma 2, there are two dom (a)-isomorphic atoms h and h' on P with dom (a)-isomorphic types. By Lemma 1, subtree (h) is dom (a)-isomorphic to subtree (h'). Thus, we can remove the node labeled with h and the path to h', obtaining a path that is at least one edge shorter. Let i be the homomorphism mapping subtree (h') to subtree(h), and let ^' = ^ o i. Then, ^' is a homomorphism that maps Q into chase(D, £) such that depth(^') < depth(^), which contradicts ^ being a homomorphism of this kind such that depth(^) is minimal. This shows that ^(Q) is contained in g-chasek(D, £). □	[redacted]
Theorem 5 Guarded TGDs enjoy the BGDP. Proof. Consider the following fact (*):	[redacted]
(*) For each atom a € g-chase*(D, £), where i ^ 0, there is a proof of a from D and £ that is contained in g-chase(7+i)^(D, £), where y is as in Lemma 3.	[redacted]
Clearly, (*) immediately implies the BGDP, because in Lemma 4 we have	[redacted]

shown that the entire query Q maps homomorphically to h(Q) C g-chasek(D, £), for some constant k, which thus implies that every a € h(Q) has a proof that is contained in g-chase(Y+i) k(D, £). We now prove (*) by induction on i ^ 0.	[redacted]
Basis: For i = 0, we have that a € D, and thus (*) obviously holds.	[redacted]
Induction: Assume that (*) holds for some i ^ 0. We now show that (*) also holds for i + 1. Let a € g-chasei+i(D, £). If a € g-chase*(D, £), then by the induction hypothesis, there exists a proof of a that is contained in g-chase(7+i)^(D, £) C g-chase(Y+i)^(i+i)(D, £). Otherwise, there is a TGD ct, and atoms ai, a2,..., as matching the body of ct, producing in one step a. Here, ai corresponds w.l.o.g. to the guard in ct. Thus, ai € g-chase*(D, £). By the induction hypothesis, there exists a proof of ai that is contained in g-chase(Y+i)^(D, £).	[redacted]
Let us now temporarily freeze g-chase(7+i) i(D, £) and consider it as a database D*. Observe that D* contains D and a complete proof of ai. Now, given that ai was obtained via a guard, a2,..., as are instances of side atoms, and thus, for j € {2,..., s}, we have that dom(aj) C dom(ai). Since ai belongs to the frozen database D*, all of its arguments are constants relative to D*. Thus, relative to D*, the atoms a2,..., as are ground, and Lemma 3 applies, and all of a2,..., as (and trivially also ai itself) have a proof in g-chase1 (D*, £). But this means that all of ai, a2,..., as have a full proof in g-chaseY+(Y+i) i(D, £), and thus the atom a, which is obtained in one step	[redacted]

from ai, a2,..., as has a full proof in g-chase Y+(Y+i)^i+i(D, £) = g-chase (Y+i)^(i+i)(D, £). □

Theorem 6 Let R be a relational schema, D be a database for R, £ be a set of guarded TGDs on R, and Q be a BCQ over R. Then, deciding D U £ |= Q is P-complete in the data complexity.

[redacted]

Proof. As for membership in P, by the proof of Theorem 5, we obtain the following polynomial decision algorithm. We first construct g-chase(ra+i)^(D, £), where n = | Q |, 5 = | R | ■ (2w)w ■ 2|R|^(2w)w, and w is the maximal arity of a predicate in R, and we then evaluate Q on g-chase(n+iH(D, £).

[redacted]

To prove hardness for P, we give a logspace reduction from the P-complete problem of deciding whether a propositional logic program with at most two body atoms in its rules logically implies a propositional atom [38]. Let L be a propositional logic program with at most two body atoms in its rules, and let p be a propositional atom. So, L is a finite set of rules of the form h ^ bi A b2, where h is a propositional atom, and each 6 is either the propositional constant true, denoted T, or a propositional atom a». Then, we define R, D, £, and Q as follows:

R = {program, query, holds};

D = {program(h, 6i, b2) | h ^ 6i A b2 € L} U {query(p)} U {holds(T)};

£ = {program(X, Y, Z) A holds(Y) A holds(Z) ^ holds(X); query(X) A holds(X) ^ q};

Q = q.

[redacted]

Observe that only D depends on L and p, while R, £, and Q are all fixed. Observe also that D can be computed

[redacted]

in logspace from L and p. It is then not difficult to verify that L logically implies p iff
D U £ |= Q. □

Theorem 7 Let R be a relational schema, D be a database for R, £ be a set of guarded TGDs on R, and Q be a Boolean atomic query over R. Then, deciding D U £ |= Q can be done in linear time in the data complexity.

Proof. By the proof of Theorem 5, we can evaluate Q on g-chase(n+i) 5(D, £), where n = | Q |, 5 =
| R | ■ (2w)w ■ 2|R| (2w^, and w is the maximal arity of a predicate in R, which can be done as follows. For every atom a € D, we construct the tree of all potential descendants in the guarded chase forest of depth up to (n + 1) ■ 5. Since | £ | is constant, every node in this tree has only a constant number of children. Thus, the tree can be constructed in constant time, and the number of applied instances of TGDs in it is constant. Hence, the union S of all applied instances of TGDs in the trees of descendants of all a € D can also be constructed in linear time. Observe now that S is a propositional logic program, and the nodes of the guarded chase forest of depth up to (n + 1) ■ 5 are all atoms that are logically implied by D U S. Let S' be obtained from S by adding all rules a ^ q such that (i) a is an atom and the label of a potential node in the guarded chase forest and (ii) Q can be homomorphically mapped to a. Clearly, S' can also be constructed in linear time. Then, D U £ = Q iff D U S' = q, where the latter can be decided in linear time [38]. In

[redacted]

[redacted]

[redacted]

summary, this shows that deciding D U £ = Q can be done in linear time in the data complexity. □

Appendix B: Proofs for Section 4

Theorem 9 Let R be a relational schema, £ be a set of TGDs on R, and Q be a BCQ over R. If £ enjoys the BDDP, then Q is FO-rewritable.

Proof. Let Yd (which depends only on Q and R) be the depth of the derivation of Q. Let ft be the maximum number of body atoms in a TGD in £. Then, every atom in the chase is generated by at most ft atoms, of which ft — 1 are side atoms. The derivation of a is therefore contained in all its ancestors; among those, there are at most fild at level 0. If we consider the whole query Q (with |Q| = n), the number of level 0-ancestors of its atoms is at most n ■ fiYd. An FO-rewriting for Q is thus constructed as follows. Take all possible sets of n ■ fiYd atoms using predicates in R and having constants from Q and (at most n ■ fiYd ■ w, where w is the maximal arity of a predicate in R) nulls as arguments. Then, considering them as a database B, compute chaseYd (B, T). Finally, whenever Q can be homomorphically mapped to chaseYd (B, T), take all atoms in B, transform the nulls into distinct variables, and make the logical conjunction 0 out of the resulting atoms. The existential closure of the logical disjunction of all such conjunctions 0 is the rewriting of Q relative to T, denoted Qs. Observe now that, for every database D for R, it holds that D |= Qs iff D U T |= Q (i.e., chase(D, T) |= Q): this is

[redacted]

[redacted]

because every conjunction in Qs corresponds to some derivation of n atoms (soundness), and every derivation of n atoms in the levels of the chase up to Yd (i.e., all those sufficient to check whether chase(D, T) |= Q) corresponds to a conjunction in Qs (completeness). □

Appendix C: Proofs for Section 6

Theorem 13 Let R be a relational schema, TT and TE be fixed sets of TGDs and EGDs on R, respectively, where TE is separable from Tt, and Tc be a fixed set of constraints on R. Let Qc be the disjunction of all Qa with a € Tc. Then:

[redacted]

(a) If deciding D U TT |= Q V Qc is feasible in polynomial time for each fixed query Q, then so is deciding D U TT U TE |= Q V Qc.

(b) If deciding D U TT |= Q V Qc is FO-rewritable for each fixed query Q, then so is deciding D U TT U TE 1= Q V Qc.

Proof. (a) Immediate by the definition of separability.

(b) Assume that Q, TT, and Qc can be rewritten into the first-order formula $ such that for each database D, it holds that D U TT |= Q V Qc iff D |= $. Now, let ^ be the disjunction of all negated EGDs -a with a € TE. Then, D U TT U TE |= Q V Qc iff D |= $ V ^. □

[redacted]

Theorem 14 Let R be a relational schema, TT and TK be sets of TGDs and keys on R, respectively, such that TK is NC with TT . Then, TK is separable from TT .

[redacted]

Proof. The proof in [30] (Lemma 3.8) uses the restricted chase (where a TGD does not fire whenever it is satisfied). We construct a somewhat different proof for the oblivious

[redacted]

chase, which is used in the present pa¬per. We do this for self-containedness, and also because it is quite instructive to see how the oblivious chase works for TGDs and NC keys. Note that, alternatively, one can just exploit the result shown in [30] (Lemma 3.8) stating that when D satisfies TK, then the restricted chase of D relative to TT U TK is equal to the restricted chase of D relative to TT alone. We observe that the latter is homomorphically equivalent to the oblivious chase chase (D, TT). Hence, chase(D, TT U Tk) does not fail, and chase(D, Tt) is a universal model of D U TT U TK .

[redacted]

We use the standard chase order adopted in this paper: whenever the chase has generated a new atom by some TGD, it applies all applicable keys (EGDs). We show that the oblivious chase converges with such an order. Let R be a relational schema, TT be a set of TGDs on R, TK be a set of NC keys on R, and D be a database for R. Assume D satisfies TK. By Definition 3, it suffices to show that

[redacted]

(1) when chasing the TGDs in TT over D using the oblivious chase, the keys in TK never lead to a hard violation, and

[redacted]

(2) for every database D and BCQ Q, it holds that chase(D, Tt) |= Q iff chase(D, TT U Tk) |= Q.

Suppose that, in the process of constructing the oblivious chase, a TGD a = $(X, Y) ^ 3Z r(X, Z) fires. Take any key K € TK for r, where K is the corresponding set of positions of relation r. Then, since the set Ho of positions in head (a) occupied by

[redacted]

universally quantified variables is not a proper superset of the set K, there are only two possible cases: (i) K = Ho: in this case, K is actually the only key that can fire! By our particular chase order, this key is immediately applied and just eliminates the new atom a generated by a, because a has fresh nulls in all positions but those of K. (ii) At least one position in K is occupied by an existentially quantified variable in head (a): then, the new fact a generated by the application of a contains a fresh null in a position in K, and therefore it cannot violate the key k. It follows that the oblivious chase only eliminates some facts generated by some TGDs, and converges to a possibly infinite fixpoint Q without ever producing a hard violation. By results of [41], the resulting chase (D, TT U Tk) is a universal model for D U TT U Tk. It is also an endomorphic image of chase (D, Tt) via the endomorphism d: chase (D, Tt) ^ chase(D, TT U Tk), where d is defined by the union of all substitutions performed by those keys that are applied. Hence, chase(D, Tt) and chase (D, TTUTk) are homomorphically equivalent, and thus satisfy the same BCQs. □

Appendix D: Proofs for Section 7

Lemma 16 Let KB be a knowledge base in DL-Lite S, S € {F, R}. Then, (a) every TGD in TKB is linear. If KB is in DL-LiteF, and TK is the set of all EGDs encoded in QKB, then (b) every EGD in TK is a key, and (c) TK is NC with TKB.

Proof. Clearly, every TGD generated from KB is linear, which already shows (a). Furthermore, every EGD

[redacted]

[redacted]

[redacted]

generated from functionality axioms in KB is obviously a key, which then shows (b). It thus only remains to prove (c). Since every key in TK is defined on a role, the only TGDs that are potentially interacting with TK are those derived from concept inclusion axioms in KB of the form B C 3P and B C 3P-, when a func¬tionality axiom (funct P) is in KB. In such cases, we have a TGD whose head is of the form EIZ pP(X, Z) or 3ZpP(Z, X), and a key of the form P(Y3, Y\), P(Y3, Y2) ^ Y\ = Y2. In both cases, (1) the set of key positions {pP[1]} is not a proper subset of the set of X-positions {pP[1]} and {pP[2]}, respectively, and

[redacted]

(2) the existentially quantified variable Z appears only once in the head of the TGD. That is, the key is non-conflicting with the two TGDs. In summary, TK is non-conflicting with TKB. □

[redacted]

Theorem 17 Let KB be a knowledge base in DL-LiteS, S € {F, R}, and let Q be a BCQ for KB. Then, Q is satisfied in KB iff DKB U TKB |= Q V QKB.

[redacted]

Proof. By Lemma 16, the set TK of all EGDs encoded in QKB is a set of keys that is non-conflicting with TKB. By Theorem 14, TK is separable from TKB. Obviously, Q is satisfied in KB iff DKB U TKB U TK U Tc 1= Q, where Tc is the set of all constraints encoded in QKB. As argued in Section 5, the latter is equivalent to DKB U TKB U TK |= Q V Qc, where Qc is the disjunction of all queries resulting from Tc. By the definition of separability (cf. Definition 3), the latter is equivalent

[redacted]

to DKB U TKB |= Q V QKB. □

Theorem 18 Let KB be a knowledge base in DL-LiteS, S € {F, R}. Then, KB is unsatisfiable iff DKB U TKB |= QKB.

[redacted]

Proof. Observe that KB is unsatisfiable iff the BCQ Q = 3X A(X) is satisfied in KB' = KB U {A C B, A C -B}, where A and B are fresh atomic concepts. By Theorem 17, the latter is equivalent to DKB/ U £KB/ |= Q V QKB/. This is in turn equivalent to DKB/ U £KB/ |= QKB/, that is, DKB U £KB |= QKB. □

Theorem 19 Datalog±± is strictly more expressive than DL-LiteF and DL-LiteR.

[redacted]

Proof. The TGD p(X) ^ q(X, X) can neither be expressed in DL-LiteF nor in DL-LiteR, since the TGDs of concept and role inclusion axioms can only project away arguments, introduce new nulls as arguments, and change the order of arguments in the predicates for atomic concepts and abstract roles, and the EGDs for functionality axioms can only produce an atom q(c, c) from q(n, c) and/or q(c, n), where n is a null, if q(c, c) was already there before. □

[redacted]

Appendix E: Proofs for Section 8

Lemma 20 Let KB be a knowledge base in DL-Lite A, and let £K be the set of all EGDs encoded in Qkb. Then, (a) every TGD in £KB is linear, (b) every EGD in £K is a key, and (c) £K is NC with £KB.

[redacted]

Proof. Obviously, every TGD generated from KB is linear or equivalent to a collection of linear TGDs, and every EGD generated from functionality axioms in KB is a key, which already proves (a) and

[redacted]

(b). As for (c), we extend the proof of Lemma 16 from DL-Lite F to DL-Lite a. As for the TGDs that are potentially interacting with the keys, DL-Lite A newly produces TGDs for role and attribute inclusion axioms Q C R and U C V, respectively, as well as for concept inclusion axioms B C C, where C may contain general concepts of the form 3Q.D with basic roles Q and general concepts D. However, by the assumption that all role and attribute functionality axioms can only be expressed on primitive roles and attributes, respectively, the keys are trivially non-conflicting with the new TGDs in the translation from DL-LiteA. Therefore, the interesting cases (i.e., those where keys are potentially conflicting with TGDs) are exactly the same as in the proof of Lemma 16, and the rest of the proof goes in the same way. □	[redacted]
Theorem 21 Let KB be a knowledge base in DL-LiteA, and let Q be a BCQ for KB. Then, Q is satisfied in KB iff Dkb U £kb |= Q V Qkb . Proof. The proof is verbally nearly identical to the proof of Theorem 17; it only differs in using Lemma 20 instead of Lemma 16. □ Theorem 22 Let KB be a knowledge base in DL-LiteA. Then, KB is unsatisfiable iff DKB U £KB |= QKB.	[redacted]
Proof. The proof is verbally nearly identical to the proof of Theorem 18; it only differs in using Theorem 21 instead of Theorem 17. □ Theorem 23 Datalog±± is strictly more expressive than DL-LiteA. Proof. The proof is verbally nearly identical to the proof of Theorem 19,	[redacted]

referring only to DL-LiteA instead of DL-Lite^. □	[redacted]
Appendix F: Proofs for Section 9 Theorem 24 Let KB be a knowledge base in DL-Lite+, and let Q be a BCQ for KB. Then, (a) Q is satisfied in KB iff DKB U TKB |= Q V QKB, and (b) KB is unsatisfiable iff DKB U TKB |= QKB.	[redacted]
Proof. We extend the proofs of Theorems 21 and 22 to DL-Lite A. Observe first that, as for role attributes, the extended translation T produces linear TGDs and keys. Furthermore, the keys also have the NC property, since (a) by the assumed restriction on DL-LiteA, all the atomic role attributes in functionality axioms (funct UR) do not occur positively in the right-hand sides of role attribute inclusion axioms, and (b) the key positions resulting from (funct UR) are not a proper subset of the X-positions in the TGD heads generated from 5(Ur), 5(Ur)-, 35(Ur), 35(Ur)-, and p(UR). The extended translation T for identification axioms (id B I\,..., In) lies slightly outside Datalog±, since the produced EGD is not really a key, but it can intuitively be considered as a key of the virtual relation R(B, I\,..., In). It also has the NC property, since by the assumed restriction on DL-LiteA, all the atomic attributes and basic roles in identification axioms do not occur positively in the right-hand sides of inclusion axioms. □	[redacted]
Theorem 25 Let KB be a knowledge base in DL-LiteF,n, DL-LiteR,n, or DL-LiteA n, and let Q be a BCQ for KB. Then, (a) Q is satisfied in KB iff DKB U TKB |= Q V QKB, and (b)	[redacted]

KB is unsatisfiable iff DKB u TKB |= QKB .	[redacted]
Proof. Immediate by Theorems 17, 18, and 24, respectively, as the only difference is that we now have multi-linear TGDs instead of linear ones. □ Theorem 26 Let KB be a knowledge base in DLR-LiteF,n, DLR-LiteR,n, or DLR-LiteA n, and let Q be a BCQ for KB. Then, (a) Q is satisfied in KB iff DKB U TKB |= Q V QKB, and (b) KB is unsatisfiable iff DKB u TKB |= QKB. Proof. Immediate by Theorem 25, since also the extended translation T produces only linear TGDs and NC keys. In particular, the NC property of keys follows from the restriction of DLR-LiteF,n, DLR-LiteR,n, and DLR-LiteA n, respectively, that all n-ary relations R in functionality axioms (funct i: R) do not appear positively in the right-hand sides of concept inclusion axioms and of the newly introduced inclusion axioms between projections of relations. □	[redacted]
Theorem 27 Datalog± is strictly more expressive than DL-LiteF,n, DL-LiteR,n, DL-LiteA n, DLR-LiteF,n, DLR-LiteR,n and DLR-Lite+ n.	[redacted]
Proof. Following the same line of argumentation as in the proof of Theorem 19, it can be shown that the TGD p(X) ^ q(X, X) cannot be expressed in any of the DLs stated in the theorem. □	[redacted]
Appendix G: Proofs for Section 10 Theorem 29 Let R be a relational schema, D be a database for R, and T be a set of guarded TGDs on R. Then, there exists an isomorphism from chase (D, T) to the least Herbrand model M of D and Tf.	[redacted]

Proof. By induction along the construction of chase(D, £), we define an isomorphism i from chase(D, £) to a subset of M as follows. For every c € A U AN that occurs in D, we define i(c) = c. Trivially, i maps D C chase(D, £) isomorphically to D C M. Consider now any step of the construction of chase(D, £), and let C be the result of the construction thus far. Suppose that i maps C C chase(D, £) isomorphically to a subset of M. Suppose that the next step in the construction of chase(D, £) is the application of the TGD ct = $(X, Y) ^ 3Z ^(X, Z), which produces the atom ^(x, N) from the atoms in $(x, y). We then extend i by mapping the vector N of nulls N € Act Z , where the Z's are the existentially quantified variables in ct, to the vector fCT (x, y) of terms /(x, y). Notice that i is injective, since every pair (x, y) uniquely determines the atoms in $(x, y) and thus at most one application of the TGD ct. Since i($(x, y)) is a subset of M, and M is a model of D and £f, also i(^(x, N)) must belong to M. Hence, i also maps the result of applying ct on C to a subset of M. We can thus construct an isomorphism i from chase(D, £) to a subset M' of M. But since M' is also a model of D and £f, as otherwise the construction of chase(D, £) would be incomplete, and since M is the least model of D and £f, we obtain M' = M. Thus, i maps chase(D, £) isomorphically to M. □	[redacted]
Proposition 30 Let R be a relational schema, D be a database for R, and £ be a stratified set of guarded normal	[redacted]

TGDs on R. Let S be a canonical model of D and £. Then, S is also a model of D and £.	[redacted]
Proof. Let a stratification of £ be given by ^: R ^ {0,1,..., k}. By induction on the stratification ^, we now show that for any construction of a canonical model S0,..., Sk, every Si is a model of D and £*. Thus, in particular, the canonical model Sk of D and £ is a model of D and ££ = £. Observe first that (*) if Sj, where j € {0,..., i} and i € {0,..., k}, is the set of all atoms a € Si such that ^(pred(a)) ^ j, then every Sj coincides with Sj.	[redacted]
Basis: Since So = chase(D, £0), and chase(D, £0) is a universal model of D and £0, in particular, S0 is a model of D and £0 = £0.	[redacted]
Induction: Suppose that Si-i is a model of D and £*-i; we now show that also Si is a model of D and £*. S. Recall first that Si = chase(Si-i, £i1-1). We have to show that (i) D can be homomorphically mapped to Si and (ii) every ct € £* is satisfied in Si. As for (i), since Si-i is a model of D and £*-i by the induction S . hypothesis, D can be homomorphically mapped to Si-i. Since Si = chase(Si-i, £i*-1) is a universal S . model Si-i and £i 1-1, it follows that Si-i can be homomorphically mapped to Si. In summary, D can be homomorphically mapped to Si. As for (ii), consider any ct € £*. Then, ct € £j for some j € {0,..., i}, S’ S’ and since (1) Sj = chase(Sj-i, £’-1) is a universal model of Sj-i and £’-1 (or	[redacted]

Sj = chase(D, £0) is a universal model of D and £0, if j = 0) and (2) Sj-i coincides with Sj-i, by (*), it follows that ct is satisfied in Sj, and since Sj coincides with Sj, by (*), it follows that ct is also satisfied in Si. □

Proposition 31 Let R be a relational schema, D be a database for R, and £ be a stratified set of guarded normal TGDs on R. Let U and V be two canonical models of D and £. Then, U is isomorphic to V.

Proof. Let a stratification of £ be given by ^: R ^ {0,1,..., k}. By induction on the stratification ^, we now show that for any two constructions of canonical models S0,..., Sk and T0,..., Tk, it holds that Si is isomorphic to Ti, for every i € {0,..., k}. Thus, in particular, the two canonical models Sk and Tk of D and £ are isomorphic.

Basis: Clearly, S0 = chase(D, £0) and T0 = chase(D, £0) are isomorphic.

Induction: Suppose that Si-1 and Ti-1 are isomorphic; we now show that also Si and Ti are isomorphic.

S' T'

Since Si-i and Ti-i are isomorphic by the induction hypothesis, Ti *-1 is isomorphic to Tiz~1. Following the construction of the chase, the isomorphism between Si-1 and Ti-1 can then be extended to an isomor¬phism between Si = chase(Si-1, TSz-1) and Ti = chase(Ti-1, Tiz-1). □

Theorem 32 Let R be a relational schema, D be a database for R, T be a stratified set of guarded normal TGDs on R, and Q be a safe normal BCQ over R. Then, there exists some

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

l ^ 0, which depends only on Q and R, such that D U T \=strat Q iff Q evaluates to true on Sk, where the sets Si,, i € {0,... ,k}, are defined as follows: (i) So = g-chase1 (D, To); (ii) if i > 0, then Si = g-chasel,Si-1 (Si-1, Ti,).	[redacted]
Proof. Let l = n ■ 5, where n = | Q+ | + 1, 5 = | R | ■ (2w)w ■ 2\n']'(2W'w, and w is the maximal arity of a predicate in R. The result is proved in the same way as Lemma 4, except that the atoms of Q may now belong to different levels of a stratification (and thus the path P of length greater than 5 for the proof by contradiction must be completely inside one level of the stratification), and one also has to check that the negative atoms (since Q is safe, their arguments are fully determined, once some candidates for the images of the positive atoms under the homomorphism are found) do not match with any of the atoms in a canonical model of D and T (which is done separately for each negative atom). □	[redacted]
Theorem 33 Let R be a relational schema, D be a database for R, T be a stratified set of guarded normal TGDs on R, and Q be a safe normal BCQ over R. Then, D U T l=strat Q is decidable in polynomial time in the data complexity.	[redacted]
Proof. The result is proved in the same way as Theorem 6, which follows from Theorem 5. The main difference is that the finite part of the guarded chase forest is now computed for each level of a stratification, and that we now also have to check that the negative atoms	[redacted]

cannot be homomorphically mapped to a canon¬ical model. We first compute a stratification of T, which is possible in constant time. We then compute sets similar to the Si’s, i € {0,... ,k}, of Theorem 32. But to obtain all side atoms, by Theorem 5, with a slightly larger depth, namely, l = (n + 1) ■ 5, where n = |Q+1 + 1, 5 = |R| ■ (2w)w ■ 2\'R]]'(2w')W, and w is the maximal arity of apredicate in R. By the proof of Theorem 6, this and the evaluation of Q+ and all Q+ U {a}, where a € Q-, over Sk is possible in polynomial time. □	[redacted]
Theorem 34 Let R be a relational schema, T be a stratified set of linear normal TGDs on R, and Q be a safe normal BCQ over R. Then, Q is FO-rewritable.	[redacted]
Proof. We use the same line of argumentation as in the proofs of Theorem 9 and Corollary 10, except that we now determine first a stratification ^ of T and then iteratively (for each possible collection of database atoms with nulls as arguments) the guarded chase forest of bounded depth (which depends only on Q and R) for every level of ^, and we also check that the negative atoms do not match with any of the atoms in the thus generated canonical model. □	[redacted]
Lemma 35 Let R be a relational schema, D be a database for R, and T be a set of guarded normal TGDs on R. Let Mf, Mf, Nf, and Nf be models of D and Tf, let M C HBs (resp., N C HBs) be an isomorphic image of Mf and Mf (resp., Nf and Nf). Then, Mf ^ Nf iff Mf ^ Nf.	[redacted]
Proof. Since M C HB s is an isomorphic image of both Mf and	[redacted]

Mf, it follows that Mf and Mf are
f f
isomorphic. Similarly, also N{ and N are isomorphic, and the two subrelations of -< that are obtained from -< by restriction to Mf x Nf and Mf x Nf are isomorphic. □

[redacted]

Proposition 36 Let R be a relational schema, D be a database for R, and £ be a set of guarded TGDs on R. Then, M is a perfect model of D and £ iff M is an isomorphic image of the least model of D and £f.

[redacted]

Proof. (^) Let M be a perfect model of D and £. That is, (i) M C HB s is an isomorphic image of a model Mf of D and £f and (ii) M ^ N for all isomorphic images N C HBs of models of D and £f such that N is not isomorphic to M. Towards a contradiction, suppose that Mf is not a minimal model of D and £f. That is, there exists a minimal model Nf of D and £f such that Nf C Mf. Thus, Mf — Nf = 0. However, since -< is empty, M ^ N does not hold, and since N is not isomorphic to M, this contradicts M being a perfect model of D and £. This shows that Mf is a minimal model of D and £f.

[redacted]

(^) Let M be an isomorphic image of the least model Mf of D and £f, and let N be any isomorphic image of a model Nf of D and £f such that N is not isomorphic to M. Since Mf C Nf, it follows that Mf — Nf = 0. Hence, since M is not isomorphic to N, it holds that M ^ N. This shows that M is a perfect model of D and £. □

[redacted]

Proposition 37 Let R be a relational schema, D be a database for R, and £ be a set of guarded nor¬mal TGDs on R with stratification ^: R ^ {0,1,..., k}. Let S C HB * and S' C

[redacted]

HB *+i such that S' n HB* = S. Then, for all i € {0,1,..., k—1}, S' is a perfect model of D*+i and £*+i iff (i) S is a perfect model of D* and £*, and S is an isomorphic image of a model Sf of D* and (£*)f, and (ii) S' is an isomorphic image of the least model of Sf U Di+i and (£f+i)s/.

Proof. Consider any i € {0,1,..., k—1}.

(^) Suppose that S' is a perfect model of D*+i and £*+i. That is, (1) S' is an isomorphic image of a model Mf of D*+i and (£*+i)f and (2) S' ^ N for all isomorphic images N C HB*+i of models of D*+i and (£*+i)f such that N is not isomorphic to S'. Hence, (1) S is an isomorphic image of a model Sf of D* and (£*)f and (2) S ^ N for all isomorphic images N C HB * of models of D* and (£*)f such that N is not isomorphic to S. That is, (i) S is a perfect model of D* and £*. Furthermore, as for (ii), since S' is an isomorphic image of a model Mf of D*+i and (£*+i)f, it follows that S' is an isomorphic image of a model Mf of Sf U Di+i and (£f+i)s/. We now show that Mf is also minimal. Towards a contradiction, suppose that Mf is not minimal. That is, there exists a minimal model Nf of Sf U Di+i and (£f+i)sf such that Nf C Mf. It thus follows that Mf — Nf = 0. Observe that Nf is also a model of D*+i and (£*+i)f. Since Mf n (HB *)f = Nf n (HB *)f, where (HB *)f is the natural extension of HB * by function symbols, S' ^ N does not hold, where N is an isomorphic image of Nf. But, since N is not isomorphic to S', this contradicts S' being a perfect model of D*+i and £*+i. This shows that Mf is also

[redacted]

[redacted]

minimal.
(^) Suppose that (i) S is a perfect model of D* and £*, and S is an isomorphic image of a model Sf of D* and (£*)f, and (ii) S' is an isomorphic image of a minimal model Mf of Sf U Di+i and (£f+i)s/. Thus, S' is also an isomorphic image of a model Mf of D*+i and (£*+i)f. We now show that S' is a perfect model of D*+i and £*+i. That is, S' ^ N for all isomorphic images N of models Nf of D*+i and (£*+i)f such that N is not isomorphic to S'. Recall that S' ^ N iff for every a € Mf — Nf, some b € Nf — Mf exists with a ^ b. Clearly, by (i), if a € (Mf — Nf) n (HB*)f, then (*) some b € (Nf — Mf) n (HB*)f exists with a ^ b. W.l.o.g., both Mf and Nf are minimal, and thus Mf n (HB*+i)f and Nf n (HB*+i)f are obtained from Sf = Mf n (HB*)f and Tf = Nf n (HB*)f, respectively, by iteratively applying an immediate consequence operator via (Tf+1)S/ and (Tf+1)Tf, respectively. Let a0,a1,... be the ordered sequence of all elements in (Mf — Nf) n (HB *+1)f such that for every i €{0,1,...}, it holds that ai is de¬rived before ai+1. Then, a0 € (Mf — Nf) n (HB *+1)f is justified either by some a € (Mf — Nf) n (HB *)f with a0 ^ a (as argued above, this implies (*)) or by some b € (Nf — Mf) n (HB*)f with a0 -< b. Similarly, every ai € (Mf — Nf) n (HB*+1)f is justified either by some aj € (Mf — Nf) n (HB*+1)f with j € {0,1, ...,i — 1} and ai ^ aj, (by induction on a0,a1,..., this implies (*) with a = aj), by some a € (Mf — Nf) n (HB*)f with ai ^ a (as argued above, this implies (*)), or by some b € (Nf —

[redacted]

Mf) n (HB*)f with aj, < b. In summary, this shows that for every a € Mf — Nf, there exists some b € Nf — Mf such that a < b. □

Theorem 39 Let R be a relational schema, D be a database for R, and T be a stratified set of guarded normal TGDs on R. Then, M is a canonical model of D and T iff M is a perfect model of D and T.

[redacted]

Proof. Let ^: R ^ {0,1,... ,k} be a stratification of T. Let S0 = chase (D, T0) and Si+1 = chase (S^ T^ 1) for i € {0,1,... ,k — 1}. Recall that Sk is the canonical model of D and T. We now show by induction on i € {0, 1,... ,k} that S-i — (Di+1 U ■ ■ ■ U Dk) is a perfect model of D* and T*. Hence, in particular, Sk is a perfect model of D* = D and T\ = T.

[redacted]

Basis: By Theorem 29, S0 — (D1 U ■ ■ ■ U Dk) is an isomorphic image of the least model of D0 and Tf. By Proposition 36, it thus follows that S0 — (D1 U ■ ■ ■ U Dk) is a perfect model of D0 = D* and T0 = T*.

[redacted]

Induction: By the induction hypothesis, Si = S,, — (Di+1 U ■ ■ ■ U Dk) is a perfect model of D* and T*, which also implies that Si is an isomorphic image of a model Sf of D* and (T*)f. By Theorem 29, Si+1 is an isomorphic image of the least model of Si and (Tf+1)Si. Thus, S'+1 = Si+1 — (Di+2 U ■ ■ ■ U Dk) is an isomorphic image of the least model of Si U Di+1 and (Tf+1)S'. Hence, S'+1 is also an isomorphic image of the least model of Sf U Di+1 and (Tf+1)S'. By Proposition 37, S^ is a perfect model of D*+1 and T*+1. □

[redacted]