Complexity of the Stable Model Semantics for Queries on Incomplete Databases

Jos de Bruijn[1] and Stijn Heymans[2]

[1] KRDB Research Center, Free University of Bozen-Bolzano, Italy
debruijn@inf.unibz.it
[2] Institute of Information Systems, Vienna University of Technology, Austria
heymans@kr.tuwien.ac.at

Abstract. We study the complexity of consistency checking and query answering on incomplete databases for languages ranging from non-recursive Datalog to disjunctive Datalog with negation under the stable model semantics. We consider both possible and certain answers and both closed- and open-world interpretation of *C-databases* with and without conditions. By reduction to stable models of logic programs we find that, under closed-world interpretation, adding negation to (disjunctive) Datalog does not increase the complexity of the considered problems for C-databases, but certain answers for databases without conditions are easier for Datalog without than with negation. Under open-world interpretation, adding negation to non-recursive Datalog already leads to undecidability, but the complexity of certain answers for negation-free queries is the same as under closed-world interpretation.

1 Introduction

In applications of relational databases a need often arises for representing incomplete information [5], typically in the form of *null values*. For example, in data exchange [8,17] anomalies in the semantics for solutions may arise if nulls are not treated with care. In data integration [1,12,16] incomplete information arises when integrating different complete data sources using a global schema; a materialized view may be incomplete with respect to local sources and local sources may be incomplete with respect to the global schema or constraints of other sources.

Null values in incomplete databases are represented using variables. A single database represents several possible instances, called *representations*. In our treatment we follow the landmark paper by Imieliński and Lipski [14], which considers both *closed-world* and *open-world* interpretation of incomplete databases. In the former, representations are in direct correspondence with valuations of the variables; each tuple in a representation is the valuation of a tuple in the database. In the latter, representations may include additional tuples not originating from the database. In addition, *local* conditions may be attached to tuples and a *global* condition to the database. Such incomplete databases with conditions are called *C-databases*.

So far, most research on query answering has been concerned with first-order and Datalog queries [25,3], and has focused mainly on data complexity. However, since those landmark papers, the formal properties of more expressive query languages such as Datalog with disjunction [7] and (unstratified) negation with the accompanying Stable Model Semantics [9] ($\mathsf{Datalog}^{\neg,\vee}$) established themselves firmly as well-accepted expressive Knowledge Representation languages. Sufficient reason for having a closer look again at those query languages for incomplete databases and thus going beyond PTime queries – queries that can be answered in polynomial time on complete databases (e.g., stratified Datalog [2]).

We study the data and combined complexity of consistency, and possible and certain answers for languages ranging from nonrecursive Datalog to $\mathsf{Datalog}^{\neg,\vee}$. We consider both the open- and closed-world interpretation of C-databases with and without conditions. Our main contributions are summarized as follows:
– We show that answering $\mathsf{Datalog}^{\neg,\vee}$ queries on incomplete databases under closed-world interpretation can be reduced to common reasoning tasks in logic programming, by an encoding of incomplete databases into logic programs.
– We present complete pictures of the data and combined complexity of consistency, and possible and certain answers for languages ranging from non-recursive Datalog to $\mathsf{Datalog}^{\neg,\vee}$, complementing earlier results [3,25] with combined complexity results for (fragments of) stratified $\mathsf{Datalog}^{\neg}$ queries and novel data and combined complexity results for queries beyond PTime. The results for closed-world interpretation are summarized in Table 1 on page 10, and for open-world interpretation in Table 2 on page 11.
– Finally, we show that results about checking uniform and strong equivalence of queries from the areas of complete databases and logic programming apply immediately to the case of incomplete databases.

To the best of our knowledge, ours are the first results about answering Datalog queries with disjunction and/or stable model negation on incomplete databases. Related to $\mathsf{Datalog}^{\neg}$ queries on incomplete databases are the techniques for consistent query answering in data integration based under the local-as-view using $\mathsf{Datalog}^{\neg}$ programs, by Bertossi and Bravo [4]. The precise relationship with query answering on incomplete databases is an open question.

Under open-world interpretation, adding negation leads to undecidability of consistency and query answering, by undecidability of finite satisfiability in first-order logic [22]. Under closed-world interpretation, for all query languages ranging from nonrecursive Datalog to $\mathsf{Datalog}^{\neg}$, the data complexity of possible answers on databases without conditions is NP-complete and of certain answers on C-databases it is coNP-complete; for certain answers coNP-completeness holds additionally for $\mathsf{Datalog}^{\vee}$ queries. For positive queries these results also apply under open-world interpretation. This shows that, for possible answers and for certain answers on general C-databases, there is no computational justification for restricting oneself to PTime languages such as Datalog and stratified $\mathsf{Datalog}^{\neg}$.

In Section 2 we review incomplete databases and define $\mathsf{Datalog}^{\neg,\vee}$ queries. In Section 3 we reduce query answering under closed-world interpretation to standard logic programming reasoning tasks. We present our complexity analysis in Section 4. Finally, we discuss related and future work in Sections 5 and 6.

An extended version of this paper, which includes the proofs of Propositions 4, 7, 10, and 11, can be found online at:
`http://www.debruijn.net/publications/sm-incomplete-db.pdf`.

2 Incomplete Databases and Queries

We consider C-databases, as defined by Imieliński and Lipski [14], and the Stable Model Semantics for logic programs, as defined by Gelfond and Lifschitz [9].

Incomplete Databases Let $\mathcal{D}$ be a countably infinite set of *constants*, called the *domain*, and let $\mathcal{V}$ be a finite set of variables, disjoint from $\mathcal{D}$. A *condition* ψ is a formula of the form $\varphi_1 \vee \cdots \vee \varphi_m$, where φ_j are conjunctions of equality atoms $x = y$ and inequality atoms $x \neq y$, with $x, y \in \mathcal{D} \cup \mathcal{V}$. A *C-table* (Conditional table) of arity n is a finite subset of $(\mathcal{D} \cup \mathcal{V})^n$ such that a *local* condition ϕ_t is associated with each tuple t in the relation. We sometimes omit ϕ_t if it is $x = x$.

A *schema* is a list $\mathcal{T} = R_1, \ldots, R_k$ of predicate symbols R_i each with an arity $n_i \geq 1$. We assume a constant bound l on the arities. A *C-database* over $\mathcal{T}$ is a tuple $\mathbf{T} = (T_1, \ldots, T_k)$ with associated condition $\Phi_{\mathbf{T}}$, such that each T_i is a C-table with arity n_i. We write individual tuples $(a_1, \ldots, a_{n_i}) \in T_i$ as $R_i(a_1, \ldots, a_{n_i})$; if $R_i(a_1, \ldots, a_{n_i})$ contains no variables, it is a *fact*. With $\mathit{preds}(\mathbf{T})$ we denote the set $\{R_1, \ldots, R_k\}$. We call a C-database *condition-free* if every condition is $x = x$. A *complete database* or *instance* I is a variable- and condition-free C-database.

Validity of variable-free conditions is defined as follows: $c_1 = c_1$ is valid; $c_1 \neq c_2$ is valid, for c_1, c_2 distinct constants; this extends to conditions in the natural way. A *valuation* is a mapping $\sigma : \mathcal{V} \cup \mathcal{D} \to \mathcal{D}$ such that $\sigma(c) = c$, for every $c \in \mathcal{D}$. This extends to tuples and conditions in the natural way. For C-tables T and C-databases $\mathbf{T}$ we define $\sigma(T) = \{\sigma(t) \mid t \in T \,\&\, \sigma(\phi_t) \text{ is valid}\}$ and $\sigma(\mathbf{T}) = (\sigma(T_1), \ldots, \sigma(T_k))$.
The *closed-world interpretation* (CWI) of a C-database $\mathbf{T}$ with arity $(n_1, \ldots, n_k)$ is defined as:

$$\mathit{rep}(\mathbf{T}) = \{\sigma(\mathbf{T}) \mid \sigma \text{ is a valuation such that } \sigma(\Phi_{\mathbf{T}}) \text{ is valid}\} \tag{1}$$

The *open-world interpretation* (OWI) of $\mathbf{T}$ is defined as:

$$\mathit{Rep}(\mathbf{T}) = \{\mathbf{R} \subseteq \mathcal{D}^{n_1} \times \cdots \times \mathcal{D}^{n_k} \mid \exists \sigma.\sigma(\Phi_{\mathbf{T}}) \text{ is valid}, \sigma(\mathbf{T}) \subseteq \mathbf{R}, \text{ and } \mathbf{R} \text{ is finite}\} \tag{2}$$

Lemma 1 (Implicit in [14]). *Let* $\mathbf{T}$ *be a C-database. Then,* $\mathit{rep}(\mathbf{T}) \subseteq \mathit{Rep}(\mathbf{T})$ *and for every* $I \in \mathit{Rep}(\mathbf{T})$ *there is an* $I' \in \mathit{rep}(\mathbf{T})$ *such that* $I' \subseteq I$*.*

$\mathsf{Datalog}^{\neg,\vee}$ *Queries Atoms* are of the form $p(a_1, \ldots, a_n)$, where the a_i's are terms and p is an n-ary predicate symbol, $n \geq 1$. *Positive literals* are atoms α and *negative literals* are negated atoms $\mathit{not}\ \alpha$. A $\mathsf{Datalog}^{\neg,\vee}$ rule r is of the form:

$$h_1 \vee \cdots \vee h_l \leftarrow b_1, \ldots, b_k \tag{3}$$

where the h_i's are atoms and the b_j's are literals, such that every variable in r occurs in some positive b_j. We call $H(r) = \{h_1, \ldots, h_l\}$ the *head* and $B(r) = \{b_1, \ldots, b_k\}$ the *body* of r. If r contains no negation, then it is a $\mathsf{Datalog}^{\vee}$ rule. If $l = 1$, then r is a $\mathsf{Datalog}^{\neg}$ rule. If r is both a $\mathsf{Datalog}^{\vee}$ and $\mathsf{Datalog}^{\neg}$ rule, then it is a $\mathsf{Datalog}$ rule.

A $\mathsf{Datalog}^{\neg,\vee}$ *program* P is a countable set of $\mathsf{Datalog}^{\neg,\vee}$ rules. $\mathsf{Datalog}^{\neg}$, $\mathsf{Datalog}^{\vee}$, and $\mathsf{Datalog}$ programs are defined analogously.

The set of predicate symbols of P, denoted $preds(P)$, is partitioned into sets of *intentional* ($int(P)$) and *extensional* ($ext(P)$) predicates such that there is no $p \in ext(P)$ in the head of any $r \in P$. We assume that each variable occurs in at most one $r \in P$.

The dependency graph of P is a directed graph $G(P) = \langle N, E \rangle$: $N = preds(P)$ and E is the smallest set that includes an edge $(p,q) \in preds(P)^2$ if there is an $r \in P$ such that p in some $h \in H(r)$ and q in some $b \in B(r)$; (p,q) is labeled *negative* if b is a a negative literal. P is *non-recursive* if $G(P)$ contains no cycles and *stratified* if $G(P)$ contains no cycles involving a negative edge. We use the prefixes nr- and st- for class of non-recursive and stratified programs.

Given a set $\Delta \subseteq \mathcal{D}$, the *grounding* of P with respect to Δ, denoted $gr_{\Delta}(P)$, is defined as the union of all substitutions of variables in P with elements of Δ.

Definition 1 (Queries). *If* X *is a class of programs, then an* X *query Q with signature $(e_1, \ldots, e_n) \to (o_1, \ldots, o_m)$ is a finite* X *program without constants and without empty rule heads such that $\{e_1, \ldots, e_n\} = ext(Q)$ are the input and $\{o_1, \ldots, o_m\} \subseteq int(Q)$ are the output predicates. Q is* well-defined *with respect to a database $\mathbf{T}$ if $ext(Q) \subseteq preds(\mathbf{T})$.*

We assume in the remainder that all queries are well-defined with respect to the database under consideration; further, let Δ be a set of constants, P a program, I an instance, and $\mathbf{I}$ a set of instances.

An *interpretation* M is a set of facts formed using predicate symbols in $preds(P)$ and constants Δ. Given a set of predicate symbols or constants Υ, with $M|_{\Upsilon}$ we denote the restriction of M to Υ.

If P is negation- and variable-free, M is a model of P if, for every $r \in P$, whenever $B(r) \subseteq M$, $H(r) \cap M \neq \emptyset$. The *reduct* $P^{M,\Delta}$ is obtained from $gr_{\Delta}(P)$ by (a) removing every rule $r \in gr_{\Delta}(P)$ such that $not\ b \in B(r)$ for some $b \in M$ and (b) removing all negative literals from the remaining rules.

M is a *stable Δ-model* of P with respect to I if $M|_{ext(P)} = I|_{ext(P)}$, M is a model of $P^{M,\Delta}$ and there is no model M' of $P^{M,\Delta}$ such that $M'|_{ext(P)} = M|_{ext(P)}$ and $M' \subsetneq M$. We leave out I if $I = \emptyset$ and Δ if $\Delta = \mathcal{D}$. We note that if Δ includes the constants in I and P, then the stable Δ-models of P with respect to I are the same as the stable models of $P \cup I$.

Example 1. Let $\Delta = \{a\}$. Consider the instance $I = \{p(a)\}$ and the program $P = \{q(x) \vee r(x) \leftarrow p(x), not\ s(x)\}$. $M = \{p(a), q(a)\}$ is a model of the reduct $P^{M,\Delta} = \{q(a) \vee r(a) \leftarrow p(a)\}$ and a stable Δ-model of P with respect to I; the other stable Δ-model is $\{p(a), r(a)\}$.

Definition 2 (Query Answers). *Let I be an instance, $\mathbf{I}$ a set of instances, and Q a* $\mathsf{Datalog}^{\neg,\vee}$ *query with signature $(e_1, \ldots, e_n) \to (o_1, \ldots, o_m)$.*

$$Q(I) = \{(M|_{\{o_1\}}, \ldots, M|_{\{o_m\}}) \mid M \textit{ is a stable model of } Q \textit{ with respect to } I\}$$

$$Q(\mathbf{I}) = \bigcup \{Q(I) \mid I \in \mathbf{I}\}$$

The closed-world interpretation of a query Q on a C-database $\mathbf{T}$ is written $Q(rep(\mathbf{T}))$ and the open-world interpretation is written $Q(Rep(\mathbf{T}))$.

3 Logic Programming Characterization of Queries under CWI

We reduce queries on incomplete databases under closed-world interpretation to logic programs with negation. Specifically, we show that there is a polynomial embedding of C-databases $\mathbf{T}$ into $\mathsf{Datalog}^{\neg}$ programs $P_{\mathbf{T}}$ such that the answers to a query Q on $\mathbf{T}$ correspond with the stable models of $Q \cup P_{\mathbf{T}}$ with respect to the output predicates $(o_1, \ldots, o_m)$.

Recall that the domain $\mathcal{D}$ is infinite, and thus there may be infinitely many valuations for a given variable in $\mathbf{T}$. The following lemma shows we need to consider only a finite subset.

Lemma 2 (Implicit in [3]). *Let Q be a $\mathsf{Datalog}^{\neg,\vee}$ query, $\mathbf{T}$ a C-database, $\Delta \subset \mathcal{D}$ include the constants in $\mathbf{T}$, and V the set of variables in $\mathbf{T}$. Then there is a set of constants $\Delta' \subset \mathcal{D}$ with cardinality $|V|$ such that $\Delta \cap \Delta' = \emptyset$ and.*

$$Q(rep(\mathbf{T}))|_{\Delta} = \{I|_{\Delta} \mid \sigma : V \to \Delta \cup \Delta',\ I \in Q(\sigma(\mathbf{T})), \text{ and } \sigma(\Phi_{\mathbf{T}}) \text{ is valid}\}$$

Note that for given Q, $\mathbf{T}$, Δ, and V, such a Δ' is finite, since $|V|$ is finite.

Definition 3. *Let $\mathbf{T}$ be a C-database and $\Delta \subset \mathcal{D}$ include constants in $\mathbf{T}$. For each tuple $t = R(\boldsymbol{a})$ in $\mathbf{T}$, with $\phi_t = \varphi_{t,1} \vee \cdots \vee \varphi_{t,m}$, the program $P_{\mathbf{T},\Delta}$ contains the rules*

$$R(\boldsymbol{a}) \leftarrow \varphi'_{t,i}, v_{x_1}(x_1), \ldots, v_{x_k}(x_k) \tag{4}$$

for $1 \leq i \leq m$, where $\varphi'_{t,i}$ is obtained from $\varphi_{t,i}$ by replacing '$\wedge$' with ',', and $x_1, \ldots, x_k$ are the variables occurring in t or $\varphi_{t,i}$.

$P_{\mathbf{T},\Delta}$ contains $D(c) \leftarrow$, for every $c \in \Delta \cup \Delta'$, with Δ' as in Lemma 2,

$$\begin{array}{ll} v_x(z) \leftarrow not\ v'_x(z), D(z) & \leftarrow v_x(z), v_x(y), z \neq y \\ v'_x(z) \leftarrow not\ v_x(z), D(z) & e_x \leftarrow v_x(z) \\ & \leftarrow not\ e_x \end{array} \tag{5}$$

for every variable x in $\mathbf{T}$. Finally, for $\Phi_{\mathbf{T}} = \varphi_{\mathbf{T},1} \vee \cdots \vee \varphi_{\mathbf{T},l}$, $P_{\mathbf{T},\Delta}$ contains

$$g \leftarrow \varphi'_{\mathbf{T},i}, v_{x_1}(x_1), \ldots, v_{x_k}(x_k) \qquad \leftarrow not\ g \tag{6}$$

for $1 \leq i \leq l$, where $\varphi'_{\mathbf{T},i}$ is obtained from $\varphi_{\mathbf{T},i}$ as before and $x_1, \ldots, x_k$ are the variables in $\varphi_{\mathbf{T},i}$. $P_{\mathbf{T},\Delta}$ contains no other rules.

Note that equality and inequality can be straightforwardly axiomatized using $\mathsf{Datalog}^{\neg}$ rules, such that $P_{\mathbf{T},\Delta}$ is indeed a $\mathsf{Datalog}^{\neg}$ program.

Intuitively, the rules (5) ensure the presence of an atom $v_x(c)$ in every stable model, indicating that the variable x is assigned to c. The constraints ensure that there is such a guess for each variable and this guess is unique. The rules (4) subsequently ensure evaluating the conditions.

The following proposition establishes correspondence between the answers to Q on $rep(\mathbf{T})$ and the stable models of $Q \cup P_{\mathbf{T},\Delta}$.

Proposition 1. *Let Q be a query with signature $(e_1, \ldots, e_n) \rightarrow (o_1, \ldots, o_m)$ on a C-database $\mathbf{T}$ and let Δ be a superset of the set of constants in $\mathbf{T}$. Then,*

$$Q(rep(\mathbf{T}))|_\Delta = \{(M|_{\{o_1\}}, \ldots, M|_{\{o_m\}}) \mid M \text{ is a stable model of } Q \cup P_{\mathbf{T},\Delta}\}|_\Delta$$

Proof. One can verify that M is a stable model of $P_{\mathbf{T},\Delta}$ iff $M = \sigma(\mathbf{T})$ for a $\sigma : V \rightarrow \Delta \cup \Delta'$ such that $\sigma(\Phi_\mathbf{T})$ is valid. The proposition then straightforwardly follows from the definition and Lemma 2. $\square$

Observe that the grounding of the program $P_{\mathbf{T},\Delta}$ is in general exponential in the size of $\mathbf{T}$, Δ, since the size of the non-ground rules (4) depends on the size of $\mathbf{T}$. However, we will see in Proposition 5 that using an intelligent polynomial grounding, the stable models of $P_{\mathbf{T},\Delta}$ can be computed in time NP.

Example 2. Consider a C-database $\mathbf{T}$ with ternary table T describing the flights of a plane on a particular day. $\mathbf{T}$ contains the tuples $t_1 = T(v, x_1, y_1)$ and $t_2 = T(i, x_2, y_2)$, with variables x_1, x_2, y_1, y_2, indicating that the plane flies from v to a destination x_1 with a pilot y_1 and from i to x_2 with y_2. As *mg* and *mc* are the only pilots certified to fly on i, t_1 has associated condition $x_1 \neq i \vee y_1 = mg \vee y_1 = mc$ and t_2 has condition $y_2 = mg \vee y_2 = mc$. Additionally, a pilot may not fly two stretches, hence the global condition $y_1 \neq y_2 \wedge x_1 \neq v \wedge x_2 \neq i$.

Let Δ be the set of constants in $\mathbf{T}$. Besides the guess rules (5), $P_{\mathbf{T},\Delta}$ contains

$$\begin{array}{ll}
T(v, x_1, y_1) \leftarrow x_1 \neq i, v_{x_1}(x_1), v_{y_1}(y_1) & T(i, x_2, y_2) \leftarrow y_2 = mg, v_{x_2}(x_2), v_{y_2}(y_2) \\
T(v, x_1, y_1) \leftarrow y_1 = mg, v_{x_1}(x_1), v_{y_1}(y_1) & T(i, x_2, y_2) \leftarrow y_2 = mc, v_{x_2}(x_2), v_{y_2}(y_2) \\
T(v, x_1, y_1) \leftarrow y_1 = mc, v_{x_1}(x_1), v_{y_1}(y_1) & \\
\quad g \leftarrow y_1 \neq y_2, x_1 \neq v, x_2 \neq i, v_{y_1}(y_1), v_{y_2}(y_2), v_{x_1}(x_1), v_{x_2}(x_2) & \leftarrow not\ g
\end{array}$$

Among the stable models of $P_{\mathbf{T},\Delta}$ (restricted to $preds(\mathbf{T})$) are $M_1 = \{T(v, i, mg), T(i, v, mc)\}$ and $M_2 = \{T(v, i, mc), T(i, v, mg)\}$. One can verify that these indeed correspond to elements of $rep(\mathbf{T})$. Consider the Datalog$^\neg$ query Q

$$\begin{aligned}
flying(x, y, z) &\leftarrow T(x, y, z) \\
flying(x, y, z) &\leftarrow T(x, u, z), flying(u, y, z) \\
roundtrip(z) &\leftarrow flying(x, x, z) \\
stranded(x) &\leftarrow not\ roundtrip(z), flying(x, y, z)
\end{aligned}$$

where *flying* is the transitive closure of the trips in T and the pilot is *stranded* if the departure and final destination do not coincide. The output predicate is *stranded*.

Consider the stable models M_1' and M_2' of $Q \cup P_{\mathbf{T},\Delta}$, which are extensions of M_1 and M_2, respectively. Both M_1' and M_2' contain $stranded(mg)$ and $stranded(mc)$. However, $Q \cup P_{\mathbf{T},\Delta}$ also has the stable models $\{T(v, c_{x_1}, c_{y_1}), T(i, v, mc), stranded(mc), stranded(c_{y_1})\}$ and $\{T(v, c_{x_1}, c_{y_1}), T(i, v, mg), stranded(mg), stranded(c_{y_1})\}$, and so neither $stranded(mg)$ nor $stranded(mc)$ is included in every stable model.

4 Complexity Analysis

In this section we study the complexity of checking consistency (cons) and of query answering, under the possible (poss) and certain (cert) answer semantics.

We consider two notions of complexity (cf. [23]): *combined complexity* is measured in the combined size of the database and the query and *data complexity* is measured in the size of the database – the query is considered *fixed*. We consider the following decision problems. As inputs (in parentheses) we consider a set of facts A, a C-database $\mathbf{T}$, and a query Q.

$\mathsf{cons}(\mathbf{T}, Q)$	*question:* is there an $I \in Q(rep(\mathbf{T}))$ such that $I \neq \emptyset$?
$\mathsf{poss}(A, \mathbf{T}, Q)$	*question:* is there an $I \in Q(rep(\mathbf{T}))$ such that $A \subseteq I$?
$\mathsf{cert}(A, \mathbf{T}, Q)$	*question:* for all $I \in Q(rep(\mathbf{T}))$, $A \subseteq I$?

cons^Q, poss^Q, and cert^Q are like the above except that Q is not part of the input.

We denote the consistency and certain answer problems under open-world interpretation with the symbols Cons and Cert, respectively. Their definitions are obtained from the above by replacing $rep(\cdot)$ with $Rep(\cdot)$. We do not consider possible answers in the open-world case, since representations may include facts not justified by tuples in the database.

With a problem Y (resp., Y^Q) for a class X of queries, we mean the restriction of the problem Y (resp., Y^Q) such the queries Q in the input (resp., parameter) are in the class X. We use the following notation for complexity classes: LSpace (logarithmic space), PTime, NP, coNP, $\Sigma_2^p = \text{NP}^{\text{NP}}$, $\Pi_2^p = \text{coNP}^{\text{NP}}$, PSpace, Exp (exponential time), NExp, coNExp, NExp^{NP}, and $\text{coNExp}^{\text{NP}}$. See, e.g., [6, Section 3] for definitions.

We consider the closed-world interpretation in Section 4.1 and the open-world interpretation in Section 4.2.

4.1 Complexity of Closed-World Interpretation

In order to give a full picture of the complexity, we repeat some results from literature of query answering over incomplete databases in Proposition 2 and 3. Queries in these propositions have no negation or only stratified negation such that a stable model semantics coincides with the usual minimal model semantics as defined in the respective literature. The following result is due to Abiteboul et al. [3].

Proposition 2 ([3]). *The problem poss^Q is NP-complete and cert^Q is coNP-complete for* nr-Datalog, nr-Datalog$^\neg$, Datalog, *and* st-Datalog$^\neg$ *queries.*

In addition, Grahne [11] showed that when restricting local conditions ϕ_t to conjunctions of equalities and the global condition $\Phi_\mathbf{T}$ to a conjunction of Horn clauses, the problem cert^Q can be solved in PTime for Datalog queries.

The hardness results in the following proposition follow from the hardness results for the case of complete databases; see [6,7,26]. Observe that the complement of poss for Datalog$^\vee$ queries is easily reduced to cert for stratified Datalog$^{\neg,\vee}$ queries; Π_2^p (resp., $\text{coNExp}^{\text{NP}}$)-hardness of cert^Q (resp., cert) for st-Datalog$^{\neg,\vee}$ follows immediately from Σ_2^p (resp., NExp^{NP})-hardness of poss^Q (resp., poss) for Datalog$^\vee$ queries [7]. Observe also that the problems poss and cert correspond for nr-Datalog queries on complete databases; PSpace-hardness of cert was established by Vorobyov and Voronkov [26].

Proposition 3. *The problem*
- poss^Q *is Σ_2^p-hard for* Datalog$^{\vee}$ *queries,*
- cert^Q *is Π_2^p-hard for* st-Datalog$^{\neg,\vee}$ *queries,*
- poss *and* cert *are PSpace-hard for* nr-Datalog *queries,*
- poss *is NExp^{NP}-hard for* Datalog$^{\vee}$ *queries, and*
- cert *is coNExp^{NP}-hard for* st-Datalog$^{\neg,\vee}$ *queries.*

We state our novel hardness result of combined complexity for Datalog in Proposition 4. Our novel membership results for queries beyond Datalog are in Propositions 5 and 6. The results are summarized in Table 1 on page 10.

Proposition 4. *The problem* poss *is NExp-hard and* cert *is coNExp-hard for* Datalog *queries.*

Proof (Sketch). The proof is by an encoding of nondeterministic Turing machines that run in exponential time into Datalog queries Q on C-databases $\mathbf{T}$. Q, which has one output predicate $accept$, encodes all possible transitions of the machine, using binary coding for time points and positions on the tape. $\mathbf{T}$ encodes a guess for each time point $\mathbf{j}$. Conditions ensure that in a given representation, exactly one guess is made for each time point. We have that $accept(1) \in I$ for some (resp., all) $I \in Q(rep(\mathbf{T}))$ iff some (resp., all) run(s) of T are accepting. Consequently, poss is NExp-hard and cert is coNExp-hard for Datalog queries.

The complete encoding can be found in the extended version. □

We obtain the following membership results with the help of the reduction to logic programs in Section 3 (see Proposition 1).

Proposition 5. *The problem*
- poss^Q *is in NP and* cert^Q *is in coNP for* Datalog$^{\neg}$ *queries,*
- poss^Q *is in Σ_2^p and* cert^Q *is in Π_2^p for* Datalog$^{\neg,\vee}$ *queries,*
- poss *and* cert *are in PSpace for* nr-Datalog$^{\neg}$ *queries,*
- poss *is in NExp and* cert *is in coNExp for* Datalog$^{\neg}$ *queries, and*
- poss *is in NExp^{NP} and* cert *is in coNExp^{NP} for* Datalog$^{\neg,\vee}$ *queries.*

Proof. Let $\mathbf{T}$ be a C-database, A a set of facts, Q a Datalog$^{\neg,\vee}$ query, Δ the set of constants occurring in $\mathbf{T}$ or A, V the set of variables in $\mathbf{T}$, and $P_{\mathbf{T},\Delta}$ the logic program that encodes $\mathbf{T}$ (see Definition 3). Without loss of generality we assume that the facts in A all involve output predicates of Q. By Proposition 1, $\mathsf{poss}(A, \mathbf{T}, Q)$ (resp., $\mathsf{cert}(A, \mathbf{T}, Q)$) iff for some (resp, all) stable model(s) M of $P_{\mathbf{T},\Delta} \cup Q$, $A \subseteq M$.

Consider the following algorithm for computing the stable models of $P_{\mathbf{T},\Delta} \cup Q$. Observe that for each stable model M of $P_{\mathbf{T},\Delta} \cup Q$ it must hold, by the rules (5), that (†) for each v_{x_i}, with $x_i \in V$, there is exactly one $v_{x_i}(t_{x_i}) \in M$.

1. Guess an interpretation M for $P_{\mathbf{T},\Delta} \cup Q$ such that (†) holds.
2. Check whether M is a minimal model of $(gr_{\Delta\cup\Delta'}(P'_{\mathbf{T},\Delta} \cup Q))^M$, where $P'_{\mathbf{T},\Delta}$ is obtained from $P_{\mathbf{T},\Delta}$ by replacing every $v_{x_i}(x_i)$ with $v_{x_i}(t_{x_i})$.

The size of the guess M is clearly polynomial in $\mathbf{T}$. The reduct $(gr_{\Delta\cup\Delta'}(P'_{\mathbf{T},\Delta}\cup Q))^M$ can be computed in time polynomial in the size of $\mathbf{T}$ (since every predicate has bounded arity) and exponential in the combined size of $\mathbf{T}$ and Q. Then, checking whether M is a minimal model of the reduct can be done in PTime if $Q \in \mathsf{Datalog}^{\neg}$ and with an NP oracle if $Q \in \mathsf{Datalog}^{\neg,\vee}$ (cf. [6]). The first, second, fourth, and fifth bullet follow immediately.

Finally, if Q does not contain recursion, it is not necessary to consider the complete grounding; the algorithm can consider the possible variable substitutions one at a time. This requires polynomial space; the third bullet follows from the fact that nondeterministic PSpace =PSpace. □

For determining the complexity of the certain answer semantics for $\mathsf{Datalog}^{\vee}$ queries we exploit the fact that entailment from $\mathsf{Datalog}^{\vee}$ programs corresponds to propositional consequence from its ground instantiation.

Proposition 6. *The problem* cert^Q *is in coNP and the problem* cert *is in coNExp for* $\mathsf{Datalog}^{\vee}$ *queries.*

Proof. We have that $\mathsf{cert}^Q(A,\mathbf{T})$ iff $A \subseteq M$ for every $I \in rep(\mathbf{T})$ and stable model M of $Q \cup I$, which is in turn equivalent to $gr_{\Delta}(Q\cup I) \models A$, where $\models$ is propositional consequence and Δ is the set of constants in I. The problem *there is an* $I \in rep(\mathbf{T})$ *such that* $gr_{\Delta}(Q\cup I) \not\models A$ can be decided as follows: (1) guess a valuation σ for the variables in $\mathbf{T}$ and a propositional valuation γ for the atoms in $gr_{\Delta}(Q\cup\sigma(\mathbf{T}))$ and (2) check $\sigma(\mathbf{T}) \in rep(\mathbf{T})$, $\gamma \models gr_{\Delta}(Q\cup\sigma(\mathbf{T}))$, and $\gamma \not\models A$. Clearly, the algorithm runs in NP in the size of $\mathbf{T}$ and in NExp in the combined size. It follows that cert^Q can be decided in coNP and cert in coNExp. □

We observe that poss can be straightforwardly reduced to cons, and vice versa.

Proposition 7. *There exists an LSpace reduction from* cons *(resp.,* cons^Q*) for a class of queries* X *to* poss *(resp.,* $\mathsf{poss}^{Q'}$*) for* X*, and vice versa.*

Therefore, our results for consistency correspond with those for possible answers.
When considering C-databases without conditions (called *V-databases* in [14]), Abiteboul et al. [3] showed that cert^Q is in PTime for $\mathsf{Datalog}$, while poss^Q is NP-complete for $\mathsf{nr\text{-}Datalog}$ queries and cert^Q is coNP-complete for $\mathsf{nr\text{-}Datalog}^{\neg}$ queries. We complement these results as follows.

Proposition 8. *When considering C-databases without conditions,* cert^Q *is in LSpace for* $\mathsf{nr\text{-}Datalog}$ *queries,* cert *is Exp-complete for* $\mathsf{Datalog}$ *queries, and* poss *is NExp-complete for* $\mathsf{Datalog}$ *queries.*

Proof. For deciding cert, variables in C-databases without conditions can be treated as constants (Skolemization), and so the database can be treated as if it were a complete database (implicit in [14,24]). Exp-completeness of cert for $\mathsf{Datalog}$ and membership in LSpace of cert^Q for $\mathsf{nr\text{-}Datalog}$ queries follows from the results for complete databases.
By Proposition 6, poss is in NExp. Hardness is proved by a slight modification of the proof of Proposition 4: the guess of the next computation step i at time point j is

	cons^Q	poss^Q	cert^Q	cons	poss	cert
nr-Datalog	**NP**	NP	coNP/LSpace	**PSpace**	**PSpace**	**PSpace**
nr-Datalog$^\neg$	**NP**	NP	coNP	**PSpace**	**PSpace**	**PSpace**
Datalog	**NP**	NP	coNP/P	**NExp**	**NExp**	**coNExp**/Exp
st-Datalog$^\neg$	**NP**	NP	coNP	**NExp**	**NExp**	**coNExp**
Datalog$^\neg$	**NP**	**NP**	**coNP**	**NExp**	**NExp**	**coNExp**
Datalog$^\vee$	$\mathbf{\Sigma_2^P}$	$\mathbf{\Sigma_2^P}$	**coNP**	$\mathbf{NExp^{NP}}$	$\mathbf{NExp^{NP}}$	**coNExp**
st-Datalog$^{\neg,\vee}$	$\mathbf{\Sigma_2^P}$	$\mathbf{\Sigma_2^P}$	$\mathbf{\Pi_2^P}$	$\mathbf{NExp^{NP}}$	$\mathbf{NExp^{NP}}$	$\mathbf{coNExp^{NP}}$
Datalog$^{\neg,\vee}$	$\mathbf{\Sigma_2^P}$	$\mathbf{\Sigma_2^P}$	$\mathbf{\Pi_2^P}$	$\mathbf{NExp^{NP}}$	$\mathbf{NExp^{NP}}$	$\mathbf{coNExp^{NP}}$

Table 1. Complexity results for C-databases with/without conditions under closed-world interpretation

performed using a single variable x_j, which may or may not be valuated with a valid i. This means that not all $I \in Q(rep(\mathbf{T}))$ correspond to runs, but still T has an accepting run iff there is an $I \in Q(rep(\mathbf{T}))$ such that $accept(1) \in I$. $\square$

The further complexity results for V-databases are the same as for C-databases. We note that the stated complexity results about V-databases apply even if variables may not occur twice in the database. Such V-databases are called *Codd databases* in [14].

Table 1 summarizes the complexity results for consistency and query answering under closed-world interpretation (CWI), both for databases with and without conditions (separated by the '/' symbol). Where the two cases correspond, only one complexity class is written. The results in boldface are novel. Note that all results in the table, save the LSpace result, are completeness results.

We can observe from the table that problems for query languages that are complete for (the complement of) a nondeterministic complexity class when considering complete databases (e.g., Datalog$^\neg$) do not increase in complexity when considering incomplete databases. So, for Datalog with disjunction and/or negation, answering queries on incomplete databases is not harder than answering queries on complete databases.

All considered PTime query languages jump to NP (resp., coNP) when considering data complexity and queries on C-databases. However, differences arise when considering the size of the query: for example, the combined complexity of the Datalog NExp-complete, whereas it is PSpace-complete for nr-Datalog.

Finally, we can observe that problems for queries on databases without conditions are only easier than those with conditions when considering PTime queries without negation and even then only certain answers are easier; possible answers and consistency are just as hard.

4.2 Complexity of Open-World Interpretation

For positive queries, certain answers under open-world interpretation (OWI) correspond to certain answers under CWI, which is a straightforward consequence of Lemma 1.

Proposition 9. *Let* $\mathbf{T}$ *be C-database,* Q *a* Datalog$^\vee$ *query, and* A *a set of facts. Then,* $\mathit{Cert}(A, \mathbf{T}, Q)$ *iff* $\mathit{cert}(A, \mathbf{T}, Q)$.

	Cons^Q	Cert^Q	Cons	Cert
nr-Datalog	**NP**/constant	coNP/LSpace	**NP/LSpace**	PSpace
nr-Datalog$^{\neg}$	Undec.	Undec.	Undec.	Undec.
Datalog	**NP**/constant	coNP/P	**NP/P**	**coNExp**/Exp
st-Datalog$^{\neg}$	Undec.	Undec.	Undec.	Undec.
Datalog$^{\neg}$	Undec.	Undec.	Undec.	Undec.
Datalog$^{\vee}$	**NP**/constant	**coNP**	$\mathbf{\Sigma_2^P}$	**coNExp**
st-Datalog$^{\neg,\vee}$	Undec.	Undec.	Undec.	Undec.
Datalog$^{\neg,\vee}$	Undec.	Undec.	Undec.	Undec.

Table 2. Complexity results for C-databases with/without conditions under open-world interpretation

Checking $\mathsf{Cons}(\mathbf{T}, Q)$ for consistent $\mathbf{T}$ (i.e., $Rep(\mathbf{T}) \neq \emptyset$) corresponds to checking satisfiability of Q, which is known to be decidable for $Q \in$ Datalog [2, Theorem 12.5.2]. Observe that databases without conditions are trivially consistent. We establish the complexity of Cons^Q and Cons in the following two propositions.

Proposition 10. *Satisfiability of* Datalog *queries is PTime-hard and satisfiability of* Datalog$^{\vee}$ *queries is Σ_2^p-hard.*

Proposition 11. *The problems Cons^Q and Cons are NP-complete for* nr-Datalog *and* Datalog*; Cons^Q is NP-complete and Cons is in Σ_2^p for* Datalog$^{\vee}$ *queries.*
When considering C-databases without conditions, Cons is in LSpace for nr-Datalog *queries and in PTime for* Datalog *queries.*

Adding negation to any of the considered query languages results in undecidability, by the undecidability of finite satisfiability of nr-Datalog$^{\neg}$ queries [22]. Table 2 summarizes the complexity results under OWI where "Undec." is short for "Undecidable" and "constant" means "decidable in constant time". All results, save the two LSpace results, are completeness results.

As can be seen from the table, checking consistency under OWI is often easier than checking consistency under CWI. Intuitively, this is the case because under CWI one needs to take the absence of tuples in the database into account.

5 Related Work

Variations on Query Languages Reiter [18] devised an algorithm for evaluating certain answers to queries on logical databases, which are essentially condition-free C-databases under CWI. The algorithm, based on relational algebra, is complete for positive first-order queries (i.e., nr-Datalog) and for conjunctive queries extended with negation in front of atomic formulas (i.e., a subsef of st-Datalog$^{\neg}$). We obtain sound and complete reasoning for free by our translation of queries on C-databases to calculating the stable models of a logic program.

Rosati [19] considers condition- and variable-free databases under OWI and certain answers for conjunctive queries and unions of conjunctive queries, as well as extensions with inequality and negation. The data complexity of such queries is polynomial as long as the queries are safe, but becomes undecidable when considering unions of conjunctive queries extended with negation involving universally quantified variables.

We considered $\mathsf{nr\text{-}Datalog}$, which generalize (unions of) conjunctive queries, but did not (yet) consider extensions with inequality and restricted forms of negation. A topic for future work is query answering on C-databases for such languages, both under CWI and OWI.

Logic Programming with Open Domains One traditionally assumes in Logic Programming information regarding individuals is complete. Hence, the grounding of logic programs with the constants in the program. Approaches that allow for *incomplete information* in that sense, e.g., where one does not need all relevant constants in the program to deduce correct satisfiability results, are, the finite k-belief sets of [10,20,21] and its generalization[3], *open answer sets* [13]. Both deal with incomplete information by not a priori assuming that all relevant constants are present in the program under consideration. It is not clear what the exact relation with C-databases is; this is part of future work.

Or-sets An alternative way of representing incomplete information is through objects with *or-sets* [15]. For example, a tuple $(\mathit{John}, \{30, 31\})$ indicates that John has age 30 or 31. This notion of incompleteness (which assumes closed-world interpretation) is somewhat simpler than C-databases and could be simulated using disjunctions of equality atoms. In [15], one shows that certain answers for existentially quantified conjunctive first-order formulas are data complete for coNP. A result that conforms with the coNP result for cert^Q with $\mathsf{nr\text{-}Datalog}$ queries in Table 1.

6 Outlook

We studied query languages ranging from $\mathsf{nr\text{-}Datalog}$ to $\mathsf{Datalog}^{\neg,\vee}$. Besides extensions of positive query languages (including conjunctive queries) with inequality and limited forms of negation (e.g., only in front of extensional predicates), in future work we plan to consider integrity constraints, both as part of the query language, as is common in logic programming, and as part of the database. While under OWI adding integrity constraints to the database leads to undecidability already for very simple query languages [19], query answering under integrity constraints for databases under CWI is largely uncharted territory. We suspect that there are cases that are undecidable under OWI, but solvable under CWI, because by Lemma 2 we need to consider only a finite subset of $\mathit{rep}(\mathbf{T})$. We note that Vardi [24] showed that checking integrity of an incomplete database is often harder under CWI than under OWI.

Abiteboul and Duschka [1] argue that a materialized view (e.g., the result of data integration) should be seen as an incomplete database, where the source predicates are seen as incomplete. Indeed, viewing a global schema as a sound view – essentially a condition- and variable-free incomplete database – is common in data integration [12,16]. Considering variables and, possibly, also conditions in (materialized) global views is a natural extension in this scenario; for example, local relations may have fewer columns than global relations, requiring view definitions of the form $\exists Y.v(X, Y, Z) \leftarrow s(X, Z)$. In future work we intend to consider query answering using such views.

[3] Both finite and infinite open answer sets are allowed.

Acknowledgements We thank the anonymous reviewers for useful comments and feedback. The work in this paper was partially supported by the European Commission under the project ONTORULE (IST-2009-231875).

References

1. S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proc. PODS*, pp. 254–263, 1998.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3. S. Abiteboul, P. C. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):158–187, 1991.
4. L. E. Bertossi and L. Bravo. Consistent query answers in virtual data integration systems. In *Inconsistency Tolerance*, LNCS 3300:42–83, 2005.
5. E. F. Codd. Extending the database relational model to capture more meaning. *ACM ToDS*, 4(4):397–434, 1979.
6. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
7. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM ToDS*, 22(3):364–418, 1997.
8. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
9. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–386, 1991.
10. M. Gelfond and H. Przymusinska. Reasoning on open domains. In *Proc. LPNMR*, pp. 397–413, 1993.
11. G. Grahne. Horn tables - an efficient tool for handling incomplete information in databases. In *Proc. PODS*, pp. 75–82, 1989.
12. A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.
13. S. Heymans, D. V. Nieuwenborgh, and D. Vermeir. Open answer set programming with guarded programs. *ACM ToCL*, 9(4), 2008.
14. T. Imieliński and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.
15. T. Imieliński, S. A. Naqvi, and K. V. Vadaparty. Incomplete objects - a data model for design and planning applications. In *Proc. SIGMOD*, pp. 288–297, 1991.
16. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pp. 233–246, 2002.
17. L. Libkin. Data exchange and incomplete information. In *Proc. PODS*, pp. 60–69, 2006.
18. R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *Journal of the ACM*, 33(2):349–370, 1986.
19. R. Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. In *Proc. PODS*, pp. 356–365, 2006.
20. J. Schlipf. Some Remarks on Computability and Open Domain Semantics. In *Proc. WS on Structural Complexity and Recursion-Theoretic Methods in Logic Programming*, 1993.
21. J. Schlipf. Complexity and Undecidability Results for Logic Programming. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):257–288, 1995.
22. B. Trakhtenbrot. Impossibility of an algorithm for the decision problem for finite models. *Dokl. Akad. Nauk SSSR*, 70:596–572, 1950.
23. M. Y. Vardi. The complexity of relational query languages (extended abstract). In *ACM Symposium on Theory of Computing*, pp. 137–146, 1982.
24. M. Y. Vardi. On the integrity of databases with incomplete information. In *Proc. PODS*, pp. 252–266, 1986.
25. M. Y. Vardi. Querying logical databases. *Journal of Computer and System Sciences*, 33(2):142–160, 1986.
26. S. G. Vorobyov and A. Voronkov. Complexity of nonrecursive logic programs with complex values. In *PODS*, pp. 244–253, 1998.