ELSEVIER

Theoretical Computer Science 193 (1998) 215–244

Theoretical Computer Science

Circumscribing DATALOG: expressive power and complexity

Marco Cadoli[a,*], Luigi Palopoli[b]

[a] *Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, I-00198 Roma, Italy*

[b] *Dipartimento di Elettronica Informatica e Sistemistica, Università della Calabria, I-87036 Rende (CS), Italy*

Received September 1995
Communicated by G. Levi

Abstract

In this paper we study a generalization of DATALOG, the language of function-free definite clauses. It is known that standard DATALOG semantics (i.e., least Herbrand model semantics) can be obtained by regarding programs as theories to be circumscribed with all predicates to be minimized. The extension proposed here, called DATALOG^{CIRC}, consists in considering the general form of circumscription, where some predicates are minimized, some predicates are fixed, and some vary. We study the complexity and the expressive power of the language thus obtained. We show that this language (and, actually, its non-recursive fragment) is capable of expressing all the queries in DB-co-NP and, as such, is much more powerful than standard DATALOG, whose expressive power is limited to a strict subset of PTIME queries. Both data and combined complexities of answering DATALOG^{CIRC} queries are studied. Data complexity is proved to be co-NP-complete. Combined complexity is shown to be in general hard for co-NE and complete for co-NE in the case of Herbrand bases containing k distinct constant symbols, where k is bounded.

1. Introduction

1.1. Background

The issue of providing a sound logic basis to knowledge representation formalisms has been, in recent years, one of the most challenging tasks for researchers. Logic-based languages, in particular, have been studied and proposed as the most natural candidates for knowledge based applications. Probably the simplest and semantically cleanest logic language for databases is DATALOG (cf. [38]), the language of universally quantified function-free definite clauses. DATALOG relies on two points:

* Corresponding author. Tel.: ++39 6 49918326; Fax: ++39 6 85300849; e-mail: cadoli@dis.uniroma1.it.

0304-3975/98/$19.00 © 1998 — Elsevier Science B.V. All rights reserved
PII S0304-3975(97)00108-4

(1) Distinction among extensional predicates (i.e., those corresponding to relations in the input database) and intensional ones (i.e., those that occur in the head of at least one rule);
(2) Uniqueness of the *least* (or *minimum*) Herbrand model of a set of rules plus a relational database.

As an example, (1) corresponds to the fact that in the following DATALOG program π:

$$non\_3\_col \leftarrow edge(X,Y), red(X), red(Y). \quad (1)$$

$$non\_3\_col \leftarrow edge(X,Y), blue(X), blue(Y). \quad (2)$$

$$non\_3\_col \leftarrow edge(X,Y), green(X), green(Y). \quad (3)$$

edge, *red*, *blue* and *green* must be extensional relations, while *non_3_col* is an intensional one.

A DATALOG program must be evaluated on a database which provides a (possibly empty) extension for each of its extensional relations. As an example, π could be evaluated on the following database D.

EDGE

FROM	TO
1	2
1	4
2	3

RED

NODE
2
4

GREEN

NODE
1

BLUE

NODE
3

According to (2) above, the intended meaning of the program is the least Herbrand model of $\pi \wedge T_D$, where T_D is the "translation" of the database D into a first-order ground formula, i.e., $edge(1,2) \wedge edge(1,4) \wedge edge(2,3) \wedge red(2) \wedge red(4) \wedge green(1) \wedge blue(3)$.

The intuitive meaning of the program π is to check whether the relations RED, BLUE and GREEN constitute a 3-coloring of the graph represented by relation EDGE. In this specific case the propositional atom *non_3_col* is false in the least Herbrand model M of $\pi \wedge T_D$, hence we know that RED, BLUE and GREEN constitute such a 3-coloring.

Semantics of DATALOG is therefore based on the declarative idea of the minimum model, i.e., of the model in which the extension of all relations is as small as possible.

1.2. The idea

In the present paper we speculate about the possibility of providing a semantics to DATALOG programs in a conceptually similar way, by employing *circumscription*, a formal system whose semantics is based on the syntax-independent idea of minimal models, which is a generalization of the idea seen above. Circumscription [27, 24] is a popular non-monotonic formalism that has been designed for the purpose of capturing some aspects of common-sense reasoning. Indeed, with circumscription, the intuitive meaning of sentences such as "normally students are young" is captured by logical formulae in which the extension of some "abnormality" predicate is minimized, while the extension of other predicates is treated differently.

For instance, consider the theory T_{mike} obtained as the conjunction of the following two clauses:

$$(\forall X)(young(X) \vee \neg student(X) \vee abnormal(X))$$
$$student(mike).$$

The intuitive meaning we would like to ascribe to the above theory is that a student is young unless he/she is explicitly known to be abnormal from this point of view. As no explicit evidence that Mike is abnormal is included in T_{mike}, Mike is young. The ordinary notion of minimization delivers two minimal Herbrand models of T_{mike}: $M_1 = \{student(mike), young(mike)\}$ and $M_2 = \{student(mike), abnormal(mike)\}$. However, M_2 does not comply with the intuitive meaning we wanted to associate with T_{mike}.

The approach of circumscription to this kind of problems is to partition the set of predicates into three subsets, of which only one – usually denoted by P – is minimized. In the above example abnormalities are to be minimized, hence *abnormal* is in P. There are predicates that we do not want to minimize, but rather to be not affected by minimization. This could be the case for *student* in the above example. Such predicates are called *fixed*, and denoted by Q. Finally, if we do not have any special attitude with respect to a property represented by a predicate – e.g., *young* – we put it in the set Z of *varying* predicates. This partition of predicates into subsets modifies the precedence relation among models: Two models are compared only if, for any given fixed predicate, they encode the same extension, and are tested for minimality w.r.t. the extensions they associate with minimized predicates. In our case, for the theory T_{mike}, the only minimal model in this new sense is M_1, which correctly captures its intended meaning.

This notion of minimality represents a generalization of classical least model semantics for DATALOG programs. Indeed, the least model semantics of DATALOG rules is obtained by circumscribing them with all the predicates to be minimized (i.e., $Q = Z = \emptyset$).

We propose to associate a semantics to DATALOG programs by employing general circumscription. What is, in the context of databases, the intuitive meaning of considering a relation to be in a set of non-minimized predicates? Referring to the initial example, we give intuitions for the set Q of fixed predicates, leaving to the body of the paper discussion on varying ones. Suppose we do not know the extension of RED, BLUE and GREEN as we do in D, but, on the contrary, we want the extension of a relation of this kind to be *arbitrary*. Then every subset of the Herbrand base $\{1,2,3,4\}$ is a possible extension for them, i.e., they become "free" relations. As a consequence, also the intensional predicates depending on the "free" ones (through rules) will have several possible different extensions associated to them. This idea of assigning arbitrary extensions to input relations has some similarities with Sagiv's notion of *uniformity* [32], which individuates properties of Datalog programs that hold independently of the input database. There are two major differences between Sagiv's notion and ours. The first one is that we use the idea of arbitrary extension to define an

alternative way to provide semantics to Datalog programs, rather than discussing about their properties. The second one is that, in our approach, only a subset of input relations have associated an arbitrary extension (e.g., the extension of the relation EDGE is not arbitrary).

As there are many possible extensions for "free" predicates and for intensional predicates depending on them, this semantics seems to be intrinsically non-deterministic. There are two ways to return to determinism:

Brave reasoning: Assume that a conclusion is valid if it holds in *at least one* possible extension;

Skeptical reasoning: Assume that a conclusion is valid if it holds in *all* possible extensions.

Brave and skeptical reasoning are often referred to also as *possibility* and *certainty* inference in the literature. As we will ground the semantics of the language on circumscription – which is intrinsically skeptical – in this paper we opt for the second possibility. Since we adopt the skeptical point of view, then the intuitive meaning of program π is the following: If for all possible choices of RED, BLUE and GREEN there is an edge whose nodes are colored the same way, then *non_3_col* becomes true. Therefore *non_3_col* will be the flag to decide whether a graph represented with relation EDGE is not 3-colorable. The intuition is formally captured minimizing only predicates *non_3_col* and *edge*, and defining Q as $\{red, green, blue\}$.

This shift from the standard least model semantics of DATALOG programs is not so exotic as it may appear. Indeed, general circumscription and logic programming have much in common, as the former has been widely used for ascribing semantics to negation. It is well known that predicate completion in Horn formulae can be partially characterized by some form of circumscription [30]. Moreover the semantics of programs with stratified negation can be given in terms of prioritized circumscription [25, 29]. Also, several semantics for non-stratified programs define intended models which are models of a circumscription (e.g., stable models [17], perfect models [29]). Finally, some forms of the negation as failure rule have been given a circumscriptive semantics in the context of closed world reasoning (e.g., careful closed world assumption [18], extended closed world assumption [19]).

Dually, there have been several attempts at compiling circumscriptive formulae into logic programs. In [16] a translation from a class of circumscriptive theories with no fixed predicates into stratified logic programs is shown. More recently, in [33] a translation method from general circumscriptive theories with fixed predicates into disjunctive logic programs with stable model semantics has been proposed.

1.3. Main contributions

This paper is devoted to study the formal properties of DATALOG^{CIRC}, the language resulting from ascribing semantics to DATALOG formulae through general circumscription. We will be particularly interested in studying the impact of adopting this new semantics on the complexity and the expressive power of DATALOG. By presenting

several examples, we will show how the language can be used to express complex queries to relational databases.

We are able to show that the non-recursive fragment of DATALOG^{CIRC}, denoted NRDATALOG^{CIRC}, is as expressive as $SO_\forall$, the universal fragment of second order logic, i.e., captures all DB-co-NP queries. The same result obviously holds for the whole language DATALOG^{CIRC}. This result holds both if we assume $Z = \emptyset$ and if we make use of varying predicates (the difference being that if we impose $Z = \emptyset$ we need a slightly more complex form of queries).

It is well known from seminal papers such as [14, 26, 8, 9, 39] that expressive power of a query language is upper bounded by its data complexity. As a consequence the price we pay for the high expressiveness of NRDATALOG^{CIRC} (as well as that of DATALOG^{CIRC}) is that the data complexity – i.e., the complexity of querying as measured when the program defining the query is considered fixed and the input database varies – is co-NP-complete. We remind that the data complexity of DATALOG is polynomial, and that there are polynomial-time expressible queries that are not captured by that language [10].

We also study the combined complexity of DATALOG^{CIRC}, the complexity measure where both the database and the program defining the query are considered part of the input. We show that, under combined complexity, the task of querying NRDATALOG^{CIRC} (and DATALOG^{CIRC}) programs is in general hard for co-NE, and is complete for co-NE when the database domain has a bounded cardinality.

1.4. Related work

The expressive power of relational database query languages has been studied for many years (cf. [22, 1]). In this subsection we mainly review papers presenting results on non-monotonic query languages.

Building on classical works about complexity and expressive power of query languages [9, 39, 21], in [10] Chandra and Harel study the expressive power of DATALOG under least models semantics and of stratified DATALOG$^\neg$, i.e., the language obtained by allowing negation to occur in rule bodies, under iterated fixpoint semantics (which is the same as perfect or stable model semantics in this case). They show that the data complexity of both languages is polynomial-time, whereas their expressive power is not enough to express all polynomial-time queries. An analogous result holds for DATALOG$^\neg$ with well-founded semantics [41].

In [23], Kolaitis and Papadimitriou study the expressive power of DATALOG$^\neg$ under fixpoint semantics. They show that, under this semantics, DATALOG$^\neg$ captures DB-NP. Furthermore, they show that the data complexity for this language is NP-complete and its combined complexity is hard for co-NE. Finally, they consider an alternative polynomial-time semantics for DATALOG$^\neg$ programs based on *inflationary* fixpoints [20]. Under inflationary fixpoint semantics, DATALOG$^\neg$ is shown to capture some, but not all polynomial-time queries (in fact, it has the same expressive power as DATALOG$^\neg$ under the well-founded model semantics, cf. e.g., [1]).

Schlipf studies in [35] the expressive power of DATALOG$^\neg$ under total stable model semantics, and shows that the skeptical version of the language captures DB-co-NP (whereas the brave version captures DB-NP). Saccà studies further the expressive power of DATALOG$^\neg$ under stable model semantics [31], by considering various variants of partial stable models. He shows that some of these variants actually increase the expressive power of DATALOG$^\neg$, inducing languages whose skeptical (resp., brave) versions are capable of capturing DB-Π_2^P (resp., DB-Σ_2^P).

An obvious difference between the languages studied in papers considered above and our approach is that we do not use negation. The semantics we propose is purely syntax-independent. This fact allows a fully declarative programming style to be maintained for DATALOG^{CIRC}. This can be contrasted against the full operational nature of DATALOG$^\neg$ under fixpoint semantics of [23] and also against the (in some sense, mild form of) operationality embedded in the definition of stable model (used in [35, 31]). Also, stable and fixpoint programs are syntax-dependent, in the sense that it is not the same to have a positive literal in the head or its negation in the body. Furthermore, note that with circumscriptive semantics DB-co-NP is already captured by the recursion-free fragment of our language, whereas recursion is obviously needed for the languages discussed in [23, 35, 31] to be fully expressive.

Eiter et al. [12] studied the complexity and the expressive power of DATALOG$^\neg$ with disjunction in the heads. They show that, under brave stable model semantics, disjunctive DATALOG$^\neg$ captures DB-Σ_2^P. Analogous results are obtained by Cadoli et al., who define in [6] a query language based on the formalism of default logic and by Bonatti and Eiter [4] who study the expressive power of several non-monotonic formalisms for querying *positive disjunctive databases*, i.e., sets of positive clauses. Eiter et al. [13] proceed one step further by considering disjunctive DATALOG$^\neg$ queries evaluated under several variants of partial stable model semantics. Their work shows that, also in this case, the use of partial models can increase the expressive power of the language up to capturing the classes DB-Σ_3^P and its complementary class DB-Π_3^P.

The languages discussed above are therefore more powerful than DATALOG^{CIRC} (unless the polynomial hierarchy collapses at a sufficiently low level) – the same holds for DATALOG$^\neg$ under L-stable semantics [31]. However, this high expressivity needs either further logical constructs to be included in the language or a complex semantics based on partial models to be employed. DATALOG^{CIRC} vice-versa, achieves a good expressive power while featuring a quite simple syntactic and semantic structure.

It is interesting to note that the negation-free version of the language presented in [12] does not capture DB-Σ_2^P, as some polynomial-time queries are not expressible through it. The full expressive power is gained by adding the predicate "$\neq$" to the query language. In DATALOG^{CIRC} it is possible to "simulate" both "$=$" and "$\neq$", as it will be shown in what follows, and DB-co-NP is captured with no additional predicate.

Finally, the computational complexity of the propositional version of circumscription has been studied in [11, 7], and the expressive power of circumscription over the integers has been investigated in [34, 36].

1.5. Structure of the paper

The structure of the paper is the following. In Section 2 we present our formal framework of reference. In Section 3, we give formal definition of DATALOG^{CIRC}, study its data complexity and show some examples of its use. In Section 4 the main result about expressive power of DATALOG^{CIRC} is proved. In Section 5 we study the combined complexity of querying relational databases through DATALOG^{CIRC} programs. Finally, in Section 6 we discuss polynomial-time sub-cases and draw conclusions.

2. Preliminaries

As for syntax, we denote variables with upper case letters (e.g., X), constant symbols with lower-case letters (e.g., c), predicate symbols with strings starting with a lower-case letter (e.g., *edge*). Lists of variables or constant symbols are denoted in bold face (e.g., $\mathbf{X}$, $\mathbf{c}$).

Herbrand interpretations and models will be denoted as sets of ground atoms (those that belong to the extensions of predicates). Interpretations and models of propositional formulae are denoted as sets of letters (those which are mapped into **true**).

In the following, we use some syntactically restricted first-order formulae, which are defined as follows.

Definition 2.1 (*Restricted clauses*). A *restricted clause* is a first-order formula of the kind

$$(\exists \mathbf{X}) \neg A_1(\mathbf{X}) \vee \cdots \vee \neg A_n(\mathbf{X}) \vee e \tag{4}$$

where each A_i $(1 \leqslant 1 \leqslant n)$ is a predicate symbol, and e is a 0-ary predicate symbol, i.e., a propositional letter.

We note that formula (4) can be equivalently written as

$$((\forall \mathbf{X}) A_1(\mathbf{X}) \wedge \cdots \wedge A_n(\mathbf{X})) \rightarrow e.$$

In what follows, queries to DATALOG^{CIRC} programs will be defined either using restricted clauses or using simple literals.

2.1. Relational databases and queries

We define queries as Boolean transformations defined on relational databases:

Let U be some countable "universal" domain of constants. A relational database is a structure D of the form $(D, R_1, \ldots, R_k)$ where $D \subset U$ is a finite set of domain constants and R_i is a relation of arity a_i over D for some a_i (i.e., $R_i \subseteq D^{a_i}$). D is said to be of type $\bar{a} = (a_1, \ldots, a_k)$ [8].

Definition 2.2 (*Chandra and Harel* [8]). A (Boolean) *computable query of type* $\bar{a} \rightarrow$ *Bool* is a mapping

$$Q : \{D \mid D \text{ is of type } \bar{a}\} \rightarrow \{True, False\}$$

satisfying the following constraints:

1. Q is partial recursive;
2. Q is generic, i.e., for each bijection ρ over D, $Q(D) = Q(\rho(D))$ (in other words, constants are uninterpreted).

Having defined Boolean queries, we draw our attention to their complexity. Each query Q defines a family of databases DB_Q as follows:

$$DB_Q = \{D \mid Q(D) = True\}.$$

The expressive power of queries is related to database complexity classes, which are defined next. Let **D** be a set of databases of type $\bar{a}$. Let $\mathscr{C}$ be a Turing complexity class (e.g., NP). Then the family **D** is said *$\mathscr{C}$-recognizable* if the problem of deciding if a given database D belongs to **D** is in $\mathscr{C}$. The *database complexity class* DB-$\mathscr{C}$ is the set of all $\mathscr{C}$-recognizable database families. Equivalently, we say that a query Q is in DB-$\mathscr{C}$ if DB_Q belongs to DB-$\mathscr{C}$. Then we say that a query language $\mathscr{L}$ *captures* a database complexity class DB-$\mathscr{C}$ if for each database family **D** in DB-$\mathscr{C}$ there is an expression $\mathscr{E}$ of $\mathscr{L}$ which defines **D**.

As for the complexity of queries, this can be measured according to two measures, namely, data and combined complexity, which have been defined in [39]. Let $\mathscr{L}$ be a query language. Let $\mathscr{E}$ be an expression of $\mathscr{L}$ defining a query $Q_{\mathscr{E}}$. Let D be an input database for $Q_{\mathscr{E}}$. Then the data complexity amounts to measuring the complexity of evaluating $Q_{\mathscr{E}}$ as a function of the size of D, while $\mathscr{E}$ is fixed. The combined complexity is given by measuring the complexity of evaluating $Q_{\mathscr{E}}$ as a function of the size D *and* the size of $\mathscr{E}$. The expression complexity, which will not be used in the present paper, amounts to measuring the complexity of evaluating $Q_{\mathscr{E}}$ as a function of the size of $\mathscr{E}$, while D is fixed.

2.2. DATALOG

A DATALOG program [38] is a finite set of universally quantified function-free first-order definite formulae T, i.e., sentences of one of the forms reported below.

$$a(\mathbf{X}) \leftarrow b_1(\mathbf{X_1}), \ldots, b_n(\mathbf{X_n})$$

$$a(\mathbf{c}) \leftarrow$$

where $n \geqslant 0$, $a(\mathbf{X})$, $b_i(\mathbf{X_i})$ $(1 \leqslant i \leqslant n)$, and $a(\mathbf{c})$ are atoms. In the former clause, the list of atoms appearing on the right of the implication sign is called *body* of the clause. The atom appearing on the left is called *head* of the clause. A clause of the latter kind is called a *fact*.

An atom is *ground* if no variables occur in it. A *ground instance* of a clause in T is constructed by consistently substituting the variables of the clause with constants occurring in T. The set of clauses occurring in T is then naturally partitioned in a subset D of facts which constitute the *input database*, and a subset π of non-ground or non-atomic formulae. We assume that no constants appear in π. A DATALOG program is said *non-recursive* if for no predicate symbol p we have $p \Leftarrow p$, where for any two (not necessarily distinct) predicate symbols p and p', $p \Leftarrow p'$ if either there is a clause where p' appears in the body and p appears in the head, or there is a predicate symbol p'' such that $p \Leftarrow p''$ and $p'' \Leftarrow p'$. Such programs will be called NRDATALOG programs.

Note that any relational database D naturally corresponds to a set of facts F_D, as follows:

$$F_D = \{a(\mathbf{c}) \leftarrow \mid a \text{ is a relation in } D \text{ and } \mathbf{c} \in a\}$$

or equivalently, as also described in Section 1, to a conjunction of all the ground atomic formulae $a(\mathbf{c})$ such that a is a relation in D and $\mathbf{c} \in D$.

Next, we discuss the semantics of DATALOG programs. A (Herbrand) model M of a DATALOG program T is a set of ground atomic formulae such that for each ground instance clause C constructible from T either the head of C is in M or at least one atom in the body of C does not belong to M. The intersection of all the Herbrand models of a DATALOG program T is itself a model for T [40, 38]. This model, denoted LM_T, is called the *least* or *minimum* model of T.

The semantics of a DATALOG program relies on its least model, as specified by the following definition.

Definition 2.3 (*Boolean queries to a* DATALOG *program*). Given a fact-free DATALOG program π, an input database D and a ground clause γ, we say that γ *follows from the least model semantics of* π *and* D (*written* $\pi, D \models \gamma$) if γ is true in $LM_{\pi \cup F_D}$.

Therefore, any fact-free DATALOG program and clause γ define a Boolean query on relational databases under the least model semantics.

The expressivity of a formalism for querying finite structures under a given semantics is given by the set of queries it defines. As already specified, interesting classes of queries for classifying the expressive power of query languages are those defined by database complexity classes.

The expressive power of DATALOG under the least model semantics is then defined by the class of queries it allows to express. It is known that DATALOG programs define a strict subset of DB-P, the class of polynomial-time queries [10]. For instance, the *even* query, i.e., to tell whether the cardinality of the set of constants in the input database is even or odd, cannot be expressed using a DATALOG program under the least model semantics [10]. The non-recursive fragment of the language is even weaker. Indeed,

the *transitive closure query*[1] is expressible using DATALOG but is not expressible with NRDATALOG [3].

2.3. Minimal models

Let T be a function-free first-order theory and $(P;Q;Z)$ a partition of its predicates. A preorder among the Herbrand models of T is defined as follows.

Definition 2.4 ($(P;Q;Z)$-*minimal models, Lifschitz* [24]). Let M, N be two Herbrand models of a formula T. We write $M \leqslant_{(P;Q;Z)} N$ if:

1. predicates in Q have the same extension in N and M;
2. for each predicate $p \in P$, the extension of p in M is a subset – possibly not proper – of the extension of p in N.

An Herbrand model M is called $(P;Q;Z)$-*minimal* for T if there is no Herbrand model N of T such that $N \leqslant_{(P;Q;Z)} M$ and $M \not\leqslant_{(P;Q;Z)} N$.

Restriction to Herbrand models such as in the above definition is not necessary, as long as only models with the same universe of interpretation are compared.

Syntactically, the circumscription of a first-order formula T is defined as a second-order formula (cf. [24] for details). In the same work it is proven that the models of such a formula are exactly the set of $(P;Q;Z)$-minimal models of T. So, for instance, the unique model of the circumscription of the formula T_{mike} that we saw in Section 1 (with $P=\{abnormal\}$, $Q=\{student\}$, $Z=\{young\}$) is $M_1=\{student(mike), young(mike)\}$.

3. The language and its data complexity

In the present section, we will give a formal definition of the DATALOG^{CIRC} query language and show, by means of some examples, how to use it for querying. Moreover, we will address the issue of data complexity of the language.

As already described, in order to construct a circumscriptive semantics of a program π, the set of predicate symbols occurring in π is partitioned into three subsets $(P;Q;Z)$. Predicates in P are those that we want to *minimize*; predicates in Q are those that we want to keep *fixed*; predicates in Z are those we want to vary. We call a 4-tuple $\langle \pi;P;Q;Z\rangle$ of this kind *a* DATALOG^{CIRC} *program*. In the specific case that π is non-recursive, $\langle \pi;P;Q;Z\rangle$ will be called *a* NRDATALOG^{CIRC} *program*. For the sake of clarity, we will explicitly substitute the symbol $\emptyset$ in the 4-tuple if one of the sets P, Q or Z is empty. Like in the plain DATALOG case, we consider only fact-free programs.

The semantics of a DATALOG^{CIRC} program relies on its Herbrand $(P;Q;Z)$-minimal models, as specified by the following definition.

[1] I.e., given a directed graph G encoded in a relation *edge* in the obvious way, and a tuple (a,b), tell if (a,b) is an edge belonging to the transitive closure of G.

Definition 3.1 (*Boolean queries to a* DATALOG^{CIRC} *program*). Let $\langle \pi; P; Q; Z \rangle$ be a fact-free DATALOG^{CIRC} program, D be an input database and γ be either a restricted clause or a literal. Then we say that γ *follows from the circumscriptive semantics of* $\langle \pi; P; Q; Z \rangle$ *and* D (*written* $\pi, D \models_{(P;Q;Z)} \gamma$) if γ is true in all Herbrand $(P; Q; Z)$-minimal models of $\pi \wedge D$.

It can be easily seen that DATALOG^{CIRC} constitutes a generalization of DATALOG: A DATALOG program π is just a DATALOG^{CIRC} program $\langle \pi; P; \emptyset; \emptyset \rangle$ in which both the set Q of fixed predicates and the set Z of varying predicates are empty.

In this paper the case of DATALOG^{CIRC} in which the set Z is empty, will have special importance.

We now see some examples of how we can use DATALOG^{CIRC} programs for doing useful computations, starting from the 3-colorability problem mentioned in Section 1. We present the formalization of the problem in DATALOG^{CIRC} in two steps: In the former, we won't make use of Z predicates, while in the latter we will.

Example 3.1 (*3-colorability of a graph*). An instance of the *3-colorability* problem (3COL) [15] is as follows. The input is a graph $G = (V, E)$. The question is: Is it possible to assign to each node in V one out of three colors, say red, blue and green, such that for each edge $e \in E$, the vertices of e have different colors?

Now, consider the following NRDATALOG^{CIRC} program $\langle \pi_{co3COL}; P; Q; \emptyset \rangle$ (note that the first three rules are exactly rules (1)–(3) shown in Section 1):

$$non\_3\_col \leftarrow edge(X, Y), red(X), red(Y).$$

$$non\_3\_col \leftarrow edge(X, Y), blue(X), blue(Y).$$

$$non\_3\_col \leftarrow edge(X, Y), green(X), green(Y).$$

$$hascolor(X) \leftarrow red(X).$$

$$hascolor(X) \leftarrow blue(X).$$

$$hascolor(X) \leftarrow green(X).$$

Here $P = \{hascolor, non\_3\_col, edge\}$, $Q = \{red, green, blue\}$, and $Z = \emptyset$. The input database D is constituted by ground atomic instances of *edge*, which is a symmetric extensional predicate encoding the set E of edges of the input graph G in the obvious way. Consider next the query γ defined by the following restricted clause:

$$((\forall X) hascolor(X)) \rightarrow non\_3\_col.$$

The intended meaning of the last three rules of π_{co3COL} is to force each node to be colored in some way. The intended meaning of the antecedent of the query γ is that we are interested in $(P; Q; \emptyset)$-minimal models of $\pi_{co3COL} \wedge D$ which assign a color to each node. As a matter of fact, we don't care if a node is assigned more than one

color: If no conflict arises with such an overloading, then we can select in an arbitrary way a color for a node among those which the node is associated with.

As it can be easily verified, $\pi_{co3COL}, D \not\models_{(P;Q;\emptyset)} \gamma$ if and only if the input graph G is 3-colorable.

In the above example the relevant property of the input graph, i.e., its 3-colorability, is captured because of the special form of the query, in which quantifiers are used. It is possible to simplify the syntactic form of the query, at the expense of allowing varying predicates.

Example 3.2 (*3-colorability of a graph, continued*). The NRDATALOG^{CIRC} program $\langle \pi^Z_{co3COL}; P'; Q; Z\rangle$ contains all the rules of $\langle \pi_{co3COL}; P; Q; \emptyset\rangle$, plus the rules

$$hascolor(X) \leftarrow z. \tag{5}$$

$$p \leftarrow z, non\_3\_col. \tag{6}$$

where $P' = P \cup \{p\}$, and $Z = \{z\}$. The input database is the same as before, while the query γ' is defined by the literal:

$$\neg z.$$

The intended meaning of rule (5) is to force each node to have at least one color. We claim that $\pi^Z_{co3COL}, D \not\models_{(P';Q;Z)} \gamma'$ if and only if the input graph G is 3-colorable. A proof of the claim follows.

Proof (*only if part*). Let us assume that $\pi^Z_{co3COL}, D \not\models_{(P';Q;Z)} \gamma'$; then there is a $(P';Q;Z)$-minimal model M of $\pi^Z_{co3COL} \wedge D$ which is not a model of γ', i.e., $M \models z$. By rule (5), this implies that $M \models (\forall X) hascolor(X)$.

We assume that $M \models non\_3\_col$, and show that a contradiction arises. Because of rule (6), it follows that $M \models p$. We build a Herbrand interpretation N of $\pi^Z_{co3COL} \wedge D$ such that

- $N \models \neg z$,
- $N \models \neg p$,

and the extension of all other predicates is the same as in M. Since N is a model of rules (5) and (6), and predicates z, p occur only in such rules, it follows that N is a model of $\pi^Z_{co3COL} \wedge D$. Since $N \leqslant_{(P';Q;Z)} M$ and $M \not\leqslant_{(P';Q;Z)} N$, it follows that M is not a $(P';Q;Z)$-minimal model of $\pi^Z_{co3COL} \wedge D$, hence a contradiction arises.

At this point we know that $M \not\models non\_3\_col$. It is easy to see that M allows a 3-coloring of G (if a node has several colors, drop all but one).

Proof (*if part*). Let us assume that G is 3-colorable and let C be a 3-coloring. Let M be an Herbrand interpretation built as follows:

- for each pair of constant symbols c, d:

- $M \models edge(c,d)$ if and only if there is an edge in G between nodes corresponding to c and d;
- $M \models red(c)$ if and only if node corresponding to c is colored in red by C;
- $M \models blue(c)$ if and only if node corresponding to c is colored in blue by C;
- $M \models green(c)$ if and only if node corresponding to c is colored in green by C;
- $M \models hascolor(c)$.

• $M \models \neg non\_3\_col \wedge z \wedge \neg p$.

Clearly M is a model of $\pi^Z_{co3COL} \wedge D$ and is not a model of γ'. All we have to prove is that M is $(P';Q;Z)$-minimal. Let us assume that there is a model N of $\pi^Z_{co3COL} \wedge D$ such that $N \leqslant_{(P';Q;Z)} M$ and $M \not\leqslant_{(P';Q;Z)} N$. This means that the extension of some predicate in P' must decrease in N w.r.t. M. Since *non_3_col* and p are already false in M and *edge* cannot decrease its extension (otherwise D is not true any more), the only possibility is to decrease the extension of *hascolor*. This in turn implies that the extension of one of the three predicates *red*, *blue* or *green* decreases, but this is impossible, as they are fixed. Therefore such a model N does not exist, and M is $(P';Q;Z)$-minimal.

Taking into account that problem 3COL is NP-complete [15], the above example shows us something about the complexity of answering Boolean queries to a NRDATALOG^{CIRC} program. As in Example 3.1 only the input database D depends on the input graph G, while both the NRDATALOG^{CIRC} program π_{co3COL} and the query γ are independent on it, we infer that the *data complexity* of deciding whether $\pi, D \models_{(P;Q;\emptyset)} \gamma$ – $\langle \pi, P; Q; \emptyset \rangle$ being a NRDATALOG^{CIRC} program and γ being a restricted clause independent on the input – is co-NP-hard. From Example 3.2, we infer that the data complexity of deciding whether $\pi, D \models_{(P;Q;Z)} l$ – $\langle \pi, P; Q; Z \rangle$ being a NRDATALOG^{CIRC} program and l being a literal – is co-NP-hard. The same hardness result obviously holds for the full DATALOG^{CIRC}. It is interesting to note that the data complexity when no varying predicates are allowed and the query is just a literal, is polynomial. This can be proven using the results on complexity of propositional circumscription shown in [7]. Intuitively, varying predicates give the possibility of stating properties of sets of models, and this accounts for the higher data complexity when the query is just a literal.

Moreover we can prove that data complexity for the problem of deciding whether $\pi, D \models_{(P;Q;Z)} \delta$ – $\langle \pi, P; Q; Z \rangle$ being a DATALOG^{CIRC} program and δ being either a literal or a restricted clause – is in co-NP by using the following argument. It is possible to generate, in non-deterministic polynomial time, an Herbrand model M of $\pi \wedge D$. The answer to the Boolean query will be "yes" if and only if for all models M generated in this way, there is none that is both a $(P;Q;Z)$-minimal model of $\pi \wedge D$ and is not a model of δ. It is well known that checking whether M is a model of δ can be done in polynomial time in the size of D (remember δ is not part of the input). Also checking whether M is a $(P;Q;Z)$-minimal model of $\pi \wedge D$ can be done in polynomial time in the size of D (cf. [5]). The same argument can be obviously applied for NRDATALOG^{CIRC}.

The above considerations give us the first computational result on DATALOG^{CIRC}.

Theorem 3.1. *Let π be a* NRDATALOG^{CIRC} *(resp.,* DATALOG^{CIRC}*) program. The data complexity of deciding whether $\pi, D \models_{(P;Q;\emptyset)} \gamma$, where the input is D and γ is a restricted clause, is co-*NP*-complete.*

*The data complexity is co-*NP*-complete also for deciding whether $\pi, D \models_{(P;Q;Z)} l$, where l is a literal.*

A second aspect that we learn from Examples 3.1 and 3.2 is that DATALOG^{CIRC} programs can be used for computing *output relations*. In fact if there is a formula γ such that $\pi, D \not\models_{(P;Q;Z)} \gamma$, we know that there is a Herbrand $(P;Q;Z)$-minimal model M of $\pi \wedge D$ such that γ is false in M. A model of this kind constitutes an extension for each non-input relation. In Example 3.2 (cf. only if part), such a model provides a 3-coloring of the input graph.

We give a second example of a co-NP-complete problem solved using DATALOG^{CIRC}.

Example 3.3 (*Set splitting*). We consider the so-called *set splitting* (3SP) [15]. An instance of 3SP is as follows. The input is a collection C of subsets of a given set S. Each element of C has cardinality 3. The question is: Is there a partition of S into two subsets S_1 and S_2 such that no subset in C is entirely contained in either S_1 or S_2? The input database D stores ground atomic formulae defining the extensional predicate c with arity 3 encoding the collection C: $c(x,y,z)$ holds if and only if $\{x,y,z\} \in C$. The program $\langle \pi_{co3SP}; P; Q; \emptyset \rangle$ is as follows:

$$non\_splitable \leftarrow c(X,Y,Z), s_1(X), s_1(Y), s_1(Z).$$

$$non\_splitable \leftarrow c(X,Y,Z), s_2(X), s_2(Y), s_2(Z).$$

$$chosen(X) \leftarrow s_1(X).$$

$$chosen(X) \leftarrow s_2(X).$$

where $P = \{chosen, non\_splitable, c\}$, $Q = \{s_1, s_2\}$, and $Z = \emptyset$. The first and the second rule of π_{co3SP} disallow a generic triplet in C to be entirely included in either S_1 or S_2. Let γ denote the following restricted clause:

$$((\forall X) chosen(X)) \rightarrow non\_splitable.$$

Then: $\pi_{co3SP}, D \not\models_{(P;Q;\emptyset)} \gamma$ if and only if the input instance, encoded in D, is indeed a yes-instance for the 3SP.

Also in this case we can simplify the query by adding some extra rules and varying predicates, as we do in the following. The DATALOG^{CIRC} program $\langle \pi^{Z}_{co3SP}; P'; Q; Z \rangle$ contains all the rules of $\langle \pi_{co3SP}; P; Q; \emptyset \rangle$, plus the rules

$$chosen(X) \leftarrow z.$$

$$p \leftarrow z, non\_splitable.$$

Here $P' = P \cup \{p\}$, and $Z = \{z\}$. The input database is the same as before, while the query γ' is defined by the literal:

$$\neg z.$$

Then: $\pi^Z_{co3SP}, D \not\models_{(P';Q;Z)} \gamma'$ if and only if the input instance is a yes-instance for the 3SP.

In this section we saw two examples of computation of co-NP-complete properties of an input database. The last example we are presenting next computes the classical polynomial-time computable "even" query. Before presenting this example, some considerations about partition of the set of predicates into minimized/fixed/varying is in order. In the above examples there is a common pattern: Predicates that belong to the set Q are always in the body of a rule, and are used for "guessing" some property (the color a node is assigned to, in Example 3.1, the set where an element is placed, in Example 3.3). This is the general usage that we propose for such special predicates, as the semantics of circumscription basically considers all possible extensions for them.

In principle, every property that is to be "guessed" should be modeled as a fixed predicate. A further, informal, example of this can be found in the DATALOG^{CIRC} program solving the "even" query reported in the example below. A way to solve such a problem is to "guess" a binary relation *pair* over the database domain having some specific properties, which are imposed by appropriate axioms. All other predicates, both the input ones and the ones which are needed for specifying axioms (e.g., *hascolor* in Example 3.1), should be put in the set P, with the exception of the letter z, which is instrumental for having simple queries, and must be put in Z.

The even query mentioned above deserves some further comment. It is known that, for expressing it, a second-order formula with alternation of existential and universal first-order quantifiers, plus predicates for denoting equality and inequality, can be used. The following example shows how quantifier alternation as well as an appropriate treatment of equality can be obtained in DATALOG^{CIRC}. In the next section we prove that DATALOG^{CIRC} has the general capability of defining equality and of expressing quantifier alternation of that form.

Example 3.4 (*Even*). An instance of EVEN is as follows. The input is a domain C of objects. The question is: Is the cardinality of C even? The input database D stores ground atomic formulae defining the extensional predicate $dom/1$ encoding the domain C: $dom(X)$ holds if and only if $X \in C$.

The rationale of the following DATALOG^{CIRC} program[2] is to "guess" a relation $pair/2$ such that each constant appears in at most one tuple of *pair*. In other words, each tuple

[2] The authors gratefully acknowledge one of the anonymous referees for having suggested the adoption of the following program for solving the even query in the place of the (much more involved) one included in the original version of this paper.

in *pair* represents a subset of the domain having cardinality two, all the subsets being disjoint with one another. The answer to the even query is then affirmative if and only if the subsets encoded in *pair* actually form a partition of the domain C.

For doing this, we will make use of several predicates, whose intended meaning is briefly summarized in the following:

$eq(X,Y)$: X and Y are equal

$\overline{eq}(X,Y)$: X and Y are not equal

$\overline{\overline{eq}}(X,Y)$: make sure that either X and Y are equal or they are not

$pair(X,Y)$: X and Y are the two elements forming one of the subsets in which the domain is tentatively partitioned

$in\_pair(X)$: X belongs to some of the subsets encoded in *pair*

fail: something wrong happened (to make sure to select only relevant models)

$dom(X)$: X is in the domain

We need several rules, each one with a specific purpose, to make sure that the intended meaning of the query is correctly captured.

1. RULES FOR EQUALITY AND INEQUALITY
   (a) $eq(X,X)$.
   (b) $\overline{\overline{eq}}(X,Y) \leftarrow eq(X,Y)$.
   (c) $\overline{\overline{eq}}(X,Y) \leftarrow \overline{eq}(X,Y)$.
   (d) $fail \leftarrow eq(X,Y), \overline{eq}(X,Y)$. /* make eq and $\overline{eq}$ disjoint */
2. RELATION *pair* PARTITIONS THE DOMAIN INTO SUBSETS OF CARDINALITY 2
   (a) $fail \leftarrow pair(X,Y), pair(X,Z), \overline{eq}(Y,Z)$.
   (b) $fail \leftarrow pair(X,Y), pair(Z,Y), \overline{eq}(X,Z)$.
   (c) $fail \leftarrow pair(X,Y), pair(Y,Z)$.
   (d) $fail \leftarrow pair(X,X)$.
3. COVERING RULES (make sure that all domain constants are put in some tuple of *pair*)
   (a) $in\_pair(X) \leftarrow pair(X,Y)$.
   (b) $in\_pair(Y) \leftarrow pair(X,Y)$.

The predicates are partitioned in the following way. The set Q is $\{\overline{eq}, pair\}$, as they are to be "guessed". Everything else is minimized, i.e., in P, and there are no varying predicates. The query γ is the restricted clause

$$((\forall X) in\_pair(X)) \rightarrow fail.$$

It is then easily seen that $\pi_{even}, D \not\models_{(P;Q;\emptyset)} \gamma$ if and only if the number of distinct symbols in the input database D is even.

As in the previous cases, we can simplify the query by using extra rules and varying predicates, as we do in the following. The DATALOG^{CIRC} program $\langle \pi^Z_{even}; P'; Q; Z \rangle$

contains all the rules of $\langle \pi_{even}; P; Q; \emptyset \rangle$, plus the rules

$$in\_pair(X) \leftarrow z.$$

$$\overline{\overline{eq}}(X, Y) \leftarrow z.$$

$$p \leftarrow z, fail.$$

Here $P' = P \cup \{p\}$, and $Z = \{z\}$. The input database is the same as before, while the query γ' is defined by the literal:

$$\neg z.$$

Then: $\pi^Z_{even}, D \not\models_{(P';Q;Z)} \gamma'$ if and only if the number of distinct symbols in the input database D is even.

In Example 3.4 we needed a special treatment for equality. We remark that we are able to "simulate" inequality by means of minimization and fixed predicates. The technique that we used is completely general, and will be employed in the next section for our main result on expressiveness of DATALOG^{CIRC}.

4. The expressive power of DATALOG^{CIRC}

In Section 3 we proved that the data complexity of NRDATALOG^{CIRC} and DATALOG^{CIRC} is co-NP-complete. In this section we prove the main result about the expressiveness of NRDATALOG^{CIRC} and DATALOG^{CIRC} by showing that they capture exactly DB-co-NP, or equivalently (cf. [14]) $SO_\forall$ queries.

Even if co-NP-completeness of querying DATALOG^{CIRC} programs under the data complexity measure have been already established in the previous section, the result we present retains full interest. In fact, the expressive power of a language is not necessarily the same as its complexity. Several languages with this property are known, cf. [2, 12]. As an example, a language which does not capture NP – even if it has an underlying NP-complete problem – has been shown by Stewart in [37].

In the following, σ denotes a fixed set of relational symbols not including equality "=" and $\mathbf{S}$ denotes a list of variables ranging over relational symbols distinct from those in σ. By Fagin's theorem [14] any NP-recognizable collection $\mathbf{D}$ of finite databases over σ is defined by a second-order existential formula.

In particular, we deal with second-order formulae of the following kind:

$$(\exists \mathbf{S})(\forall \mathbf{X})(\exists \mathbf{Y})(\theta_1(\mathbf{X}, \mathbf{Y}) \vee \cdots \vee \theta_k(\mathbf{X}, \mathbf{Y})), \tag{7}$$

where $\theta_1, \ldots, \theta_k$ are conjunctions of literals involving relational symbols in σ and $\mathbf{S}$, plus relational symbol "=". Each conjunction θ_i contains the occurrence of some variables among $\mathbf{X}, \mathbf{Y}$. (The reason why we can restrict our attention to second-order formulae in the above normal form is explained in [23]). As usual, "=" is always interpreted

as "identity". The set of uninterpreted relational symbols occurring in formula (7) – i.e., $\sigma \cup \mathbf{S}$ – will be denoted either by $\mathscr{L}$ or by $\{a_1, \ldots, a_l\}$.

We illustrate a method that transforms a formula ψ of the kind (7) into a NRDATALOG^{CIRC} program $\langle \pi_\psi; P; Q; \emptyset \rangle$ and a query γ. The transformation is complicated partly because of the special treatment necessary for equality. Both π_ψ and γ use an enlarged set of relational symbols $\mathscr{L}'$ which is built as follows:

- each relational symbol $a \in \mathscr{L}$ is in $\mathscr{L}'$;
- for each relational symbol $a \in \mathscr{L}$ there are two relational symbols $\overline{a}$ and $\overline{\overline{a}}$ with the same arity as a in $\mathscr{L}'$;
- there are three binary relational symbols $eq, \overline{eq}, \overline{\overline{eq}}$ in $\mathscr{L}'$;
- there is a relational symbol t with the same arity as $\mathbf{X}$ in $\mathscr{L}'$;
- there is a 0-ary relational symbol e in $\mathscr{L}'$.

The NRDATALOG^{CIRC} program $\langle \pi_\psi; P; Q; \emptyset \rangle$ is built as follows:

1. for each conjunct

$$\theta_i(\mathbf{X}, \mathbf{Y}) = \neg w_1(\mathbf{X}, \mathbf{Y}) \wedge \cdots \wedge \neg w_n(\mathbf{X}, \mathbf{Y}) \wedge w_{n+1}(\mathbf{X}, \mathbf{Y}) \wedge \cdots \wedge w_{n+m}(\mathbf{X}, \mathbf{Y})$$

$(1 \leqslant i \leqslant k)$ in ψ there is the rule

$$t(\mathbf{X}) \leftarrow \overline{v_1}(\mathbf{X}, \mathbf{Y}), \ldots, \overline{v_n}(\mathbf{X}, \mathbf{Y}), v_{n+1}(\mathbf{X}, \mathbf{Y}), \ldots, v_{n+m}(\mathbf{X}, \mathbf{Y}) \tag{8}$$

in π_ψ, where

- $\overline{v_i}$ $(1 \leqslant i \leqslant n)$ is
  - $\overline{eq}$, if w_i is $=$,
  - $\overline{w_i}$, otherwise;
- v_{n+i} $(1 \leqslant i \leqslant m)$ is
  - eq, if w_{n+i} is $=$,
  - w_{n+i}, otherwise.

We refer to the body of rule (8) as $\overline{\theta_i}(\mathbf{X}, \mathbf{Y})$.

2. for each $a \in \mathscr{L}$ there are three rules in π_ψ:

$$\overline{\overline{a}}(\mathbf{X}) \leftarrow a(\mathbf{X}), \tag{9}$$

$$\overline{\overline{a}}(\mathbf{X}) \leftarrow \overline{a}(\mathbf{X}), \text{ and} \tag{10}$$

$$e \leftarrow a(\mathbf{X}), \overline{a}(\mathbf{X}) \tag{11}$$

3. there are four additional rules in π_ψ:

$$eq(X, X). \tag{12}$$

$$\overline{\overline{eq}}(X, Y) \leftarrow eq(X, Y), \tag{13}$$

$$\overline{\overline{eq}}(X, Y) \leftarrow \overline{eq}(X, Y), \text{ and} \tag{14}$$

$$e \leftarrow eq(X, Y), \overline{eq}(X, Y). \tag{15}$$

Relational symbols in $\mathscr{L}'$ are partitioned in the sets P, Q and Z in the following way:

- $P = \{\overline{\overline{a}} \mid a \in \mathscr{L}\} \cup \{a \mid a \in \sigma\} \cup \{eq, \overline{\overline{eq}}, t, e\}$;
- $Q = \{\overline{a} \mid a \in \mathscr{L}\} \cup \{a \mid a \in \mathbf{S}\} \cup \{\overline{eq}\}$;
- $Z = \emptyset$.

Finally, we define as query γ for the NRDATALOG^{CIRC} program π_ψ the restricted clause:

$$((\forall \mathbf{X})t(\mathbf{X}) \wedge \overline{\overline{a_1}}(\mathbf{X}) \wedge \cdots \wedge \overline{\overline{a_l}}(\mathbf{X}) \wedge \overline{\overline{eq}}(\mathbf{X})) \rightarrow e.$$

We are now ready for the main result about expressive power of NRDATALOG^{CIRC}.

Theorem 4.1. *For any* NP*-recognizable collection* $\mathbf{D}$ *of finite databases over* σ *– characterized by a formula* ψ *of the kind* (7) *– the* NRDATALOG^{CIRC} *program* $\langle \pi_\psi; P; Q; \emptyset \rangle$ *built according to the above rules is such that a database* D *is in* $\mathbf{D}$ *if and only if* $\pi_\psi, D \not\models_{(P;Q;\emptyset)} \gamma$.

Proof (*only if part*). We assume that database D is in $\mathbf{D}$ and prove that $\pi_\psi, D \not\models_{(P;Q;\emptyset)} \gamma$.

According to Fagin's theorem [14], the database D being in $\mathbf{D}$ implies that there is a list of relations Ξ (matching the list of relational variables $\mathbf{S}$) such that

$$(D, \Xi) \models (\forall \mathbf{X})(\exists \mathbf{Y})(\theta_1(\mathbf{X}, \mathbf{Y}) \vee \cdots \vee \theta_k(\mathbf{X}, \mathbf{Y})). \tag{16}$$

Using D and Ξ we build an interpretation N of all relational symbols of $\mathscr{L}'$ such that
(α) N is a model of $\pi_\psi \wedge D$;
(β) N is a $(P; Q; \emptyset)$-minimal model of $\pi_\psi \wedge D$;
(γ) $N \not\models \gamma$,
thus proving the claim.

N is built as follows:

(i: interpretation of a symbols) for each relational symbol a of $\mathscr{L}$ and each tuple $\mathbf{t}$, $N \models a(\mathbf{t})$ if and only if $(D, \Xi) \models a(\mathbf{t})$;

(ii: interpretation of $\overline{a}$ symbols) for each relational symbol a of $\mathscr{L}$ and each tuple $\mathbf{t}$, $N \models \overline{a}(\mathbf{t})$ if and only if $N \not\models a(\mathbf{t})$;

(iii: interpretation of $\overline{\overline{a}}$ symbols) for each relational symbol a of $\mathscr{L}$ and each tuple $\mathbf{t}$, $N \models \overline{\overline{a}}\mathbf{t})$;

(iv: interpretation of $eq, \overline{eq}, \overline{\overline{eq}}$) for each pair of constant symbols c, d:

- $M \models eq(c, d)$ if and only if $c = d$;
- $M \models \overline{eq}(c, d)$ if and only if $c \neq d$;
- $M \models \overline{\overline{eq}}(c, d)$;

(v: interpretation of t) for each tuple $\mathbf{t}$, $N \models t(\mathbf{t})$;

(vi: interpretation of e) $N \not\models e$.

Proof. (α) N is a model of D because of clause (i) above. As for rules of π_ψ of the kind (8), they are satisfied by N because of clause (v). Rules of the kind (9) or (10) are satisfied because of clause (iii). Rules of the kind (11) are satisfied because of clauses (i), (ii). Rules (12)–(15) are satisfied because of clause (iv).

Proof. (β) First, note that the extensions of predicates in D cannot be decreased because, otherwise, D would not be satisfied any longer. Since the extension of relation e is already empty (cf. clause (vi)) and since extensions of predicates from D cannot be decreased, the only possibility for building a model of π_ψ with an extension of relations in P smaller than that of N is to eliminate some tuple $\mathbf{t}$ from the extension of a relation $\overline{\overline{a_i}}$ for at least an i ($1 \leqslant i \leqslant l$), or from relations $eq, \overline{\overline{eq}}, t$.

As for $\overline{\overline{a_i}}$ relations, clause (ii) implies that $N \models (\forall \mathbf{X})(a_i(\mathbf{X}) \vee \overline{a_i}(\mathbf{X}))$ for all i ($1 \leqslant i \leqslant l$). For decreasing the extension of $\overline{\overline{a_i}}$ it would be necessary – because of rules of kind (9) and (10) – to decrease the extension of a_i or $\overline{a_i}$, which is impossible because $\overline{a_i}$ is in Q and a_i is either in Q or belongs to D.

As for eq, it is impossible to decrease its extension because of rule (12).

As for $\overline{\overline{eq}}$ – because of rule (14) and of the above consideration – for decreasing its extension it would be necessary to decrease the extension of $\overline{eq}$, which is impossible because $\overline{eq}$ is a predicate in Q.

As for t, we reason as follows. For each tuple $\mathbf{X}$ there must be a tuple $\mathbf{Y}$ such that relation (16) is satisfied. By rules (i), (ii) and (iv), for each tuple $\mathbf{X}$ there must exist a conjunct θ_i and a tuple $\mathbf{Y}$ such that the body of the corresponding clause of the kind (8) is satisfied by N. As such a body is constituted by relations whose extension cannot be decreased – either because they are in Q or because they are eq or because they are in D – the above argument applies.

Proof. (γ) This follows from clauses (iii), (iv), (v) and (vi).

Proof (*if part*). We assume that there is a $(P; Q; \emptyset)$-minimal model N of $\pi_\psi \wedge D$ such that $N \not\models \gamma$ and build a list of relations Ξ (matching the list of relational variables $\mathbf{S}$) such that

$$(D, \Xi) \models (\forall \mathbf{X})(\exists \mathbf{Y})(\theta_1(\mathbf{X}, \mathbf{Y}) \vee \cdots \vee \theta_k(\mathbf{X}, \mathbf{Y})),$$

i.e., we prove that database D is in $\mathbf{D}$.

Fact 1. Since $N \not\models \gamma$, it holds that $N \models (\forall \mathbf{X})\overline{\overline{a_i}}(\mathbf{X})$ for all i ($1 \leqslant i \leqslant l$)), $N \models (\forall \mathbf{X})(\overline{\overline{eq}}(\mathbf{X}) \wedge t(\mathbf{X}))$, and $N \not\models e$.

Fact 2. Since N is $(P; Q; \emptyset)$-minimal and $\overline{\overline{eq}}$ only occurs in rules (13) and (14), it follows that $N \models (\forall X, Y)(eq(X, Y) \vee \overline{eq}(X, Y))$. Since $N \not\models e$, and considering rule (15), it follows that $N \not\models (\exists X, Y)eq(X, Y) \wedge \overline{eq}(X, Y)$, i.e., $N \models (\forall X, Y)\neg eq(X, Y) \vee \neg\overline{eq}(X, Y)$, which, taking into account the above result, implies $N \models (\forall X, Y)eq(X, Y) \oplus \overline{eq}(X, Y)$. In other words – taking into account rule (12) – in N eq is "genuine equality" and $\overline{eq}$ is "genuine inequality".

Fact 3. Since t is a minimized predicate, for each tuple $\mathbf{X}$ there must be a conjunct θ_i such that $N \models (\exists \mathbf{Y})\overline{\theta_i}(\mathbf{X}, \mathbf{Y})$.

Fact 4. Since N is $(P;Q;\emptyset)$-minimal and $\overline{\overline{a}}$ predicates only occur in rules of the kind (9) or (10), it follows that $N \models (\forall \mathbf{X})(a_i(\mathbf{X}) \vee \overline{a_i}(\mathbf{X}))$ for all i $(1 \leqslant i \leqslant l)$.

We now build an interpretation M of all relational symbols of $\mathscr{L}$ (remind that "=" is already interpreted).
(i) for each relational symbol a of $\mathscr{L}$ and each tuple $\mathbf{t}$, if $N \models a(\mathbf{t})$ then $M \models a(\mathbf{t})$;
(ii) for each relational symbol a of $\mathscr{L}$ and each tuple $\mathbf{t}$, if $N \models \overline{a}(\mathbf{t})$ then $M \models \neg a(\mathbf{t})$.
Using Fact 4, clauses (i) and (ii), we can see that M is a complete interpretation of all relational symbols in $\mathscr{L}$.

Since $N \models D$, and because of clause (i), $M \models D$. As a consequence, since relations in the input database D are minimized in N, M can be denoted as (D, Ξ), where Ξ is obtained by restriction of M to relational symbols of $\mathbf{S}$.

Using Fact 3, clauses (i) and (ii), we see that for each tuple $\mathbf{X}$ there must be a conjunct θ_i such that $(D, \Xi) \models (\exists \mathbf{Y})\theta_i(\mathbf{X}, \mathbf{Y})$, i.e., $(D, \Xi) \models (\forall \mathbf{X})(\exists \mathbf{Y})(\theta_1(\mathbf{X}, \mathbf{Y}) \vee \cdots \vee \theta_k(\mathbf{X}, \mathbf{Y}))$. □

In the previous section we saw that in specific cases it is possible to "simplify" the syntactic form of the query, at the expense of allowing varying predicates. Applying the same technique to the general case presented in this section, we obtain the NRDATALOG^{CIRC} program $\langle \pi_\psi^Z; P'; Q; Z \rangle$, which is built on the set of relational symbols $\mathscr{L}' \cup \{p, z\}$, and contains all the rules of $\langle \pi_\psi; P; Q; \emptyset \rangle$, plus the rules

$$t(\mathbf{X}) \leftarrow z$$

$$\overline{\overline{eq}}(X, Y) \leftarrow z$$

$$\overline{\overline{a_i}}(\mathbf{X}) \leftarrow z \quad (1 \leqslant i \leqslant l)$$

$$p \leftarrow z, e.$$

Where $P' = P \cup \{p\}$, and $Z = \{z\}$. The query γ' is defined by the literal:

$$\neg z.$$

The following corollary is easy to prove, using the same argument shown in Example 3.2.

Corollary 4.2. *For any* NP*-recognizable collection* $\mathbf{D}$ *of finite databases over* σ *– characterized by a formula* ψ *of the kind* (7) *– the* NRDATALOG^{CIRC} *program* $\langle \pi_\psi^Z; P'; Q; Z \rangle$ *built according to the above rules is such that a database* D *is in* $\mathbf{D}$ *if and only if* $\pi_\psi^Z, D \not\models_{(P';Q;Z)} \gamma'$.

Remark. The results presented in Theorem 4.1 and Corollary 4.2 obviously hold for the full language DATALOG^{CIRC}.

5. Combined complexity

In this section we establish combined complexity of NRDATALOG^{CIRC} and DATALOG^{CIRC}. Under combined complexity, both the database and the program are part of the input. We will show that, in this case, to determine if a restricted clause (or literal) γ follows from a given NRDATALOG^{CIRC} (DATALOG^{CIRC}) program $\langle \pi; P; Q; Z \rangle$ and database D is hard for co-NE. We remind that NE $= \bigcup_{c>1}$ NTIME (2^{cn}) (cf. [28]), i.e., NE is the set of all problems solvable by a non-deterministic machine in time bounded by 2^{cn}, where n is a measure of the size of the input and c is an arbitrary constant.

To prove our result, we use a technique shown in [23]. We begin by recalling the definition of *succinct* 3-*colorability* of a graph. Assume the nodes of the input graph are individuated by elements of $\{0,1\}^n$. Then, instead of presenting the input using an explicit *edge* relation, we are given a Boolean circuit with $2n$ inputs that outputs 1 if the two n-tuples given as its input denote two nodes connected by an edge, and outputs 0, otherwise. A Boolean circuit is a finite set of triplets $\{(a_i, b_i, c_i) \mid 1 \leqslant i \leqslant k\}$ such that a_i is the gate type, taken from the set $\{OR, AND, NOT, IN\}$, and b_i and c_i are indexes less than i, denoting the inputs of the gate. If $a_i = NOT$ then $b_i = c_i$ (as NOT is a unary operator). If $a_i = IN$, then we assume $b_i = c_i = 0$, as in this case the input of the gate is not another gate but comes from outside. Triplet (a_k, b_k, c_k) represents the output gate. The succinct 3-coloring problem (Succ3COL) is the following. The input is one such Boolean circuit with $2n$ inputs and one output encoding a graph. The question is: Is the graph thus presented 3-colorable? To prove our next result, we construct a reduction of Succ3COL to the problem of determining whether a clause follows from a given NRDATALOG^{CIRC} program or not, and then use the following result from [23].

Theorem 5.1 (Kolaitis and Papadimitriou [23, Lemma 2]). *Succ3COL is* NE-*complete.*

Let $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{e}$ and $\mathbf{d}$ denote the n-tuples $(X_1, \ldots, X_n)$, $(Y_1, \ldots, Y_n)$, $(c_1, \ldots, c_n)$ and $(e_1, \ldots, e_n)$, respectively. The reduction is as follows. Intuitively, given a Boolean circuit $G = \{(a_i, b_i, c_i) \mid 1 \leqslant i \leqslant k\}$ we construct from it a program π_G as follows: For each triplet $g_i = (a_i, b_i, c_i) \in G$, we define, in the program π_G, a predicate $g_i(\mathbf{X}, \mathbf{Y})$ that will hold true on every $2n$-tuple $(\mathbf{e}, \mathbf{d})$ that makes the gate g_i output 1. Therefore $g_k(\mathbf{e}, \mathbf{d})$ – the predicate corresponding to the output gate of the circuit – will hold true if and only if $(\mathbf{e}, \mathbf{d})$ denotes an edge of the input graph G. π_G is defined as follows. For each i, $1 \leqslant i \leqslant k$, if $a_i = AND$ then π_G will contain the clause:

$$g_i(\mathbf{X}, \mathbf{Y}) \leftarrow g_{b_i}(\mathbf{X}, \mathbf{Y}), g_{c_i}(\mathbf{X}, \mathbf{Y}).$$

Otherwise, if $a_i = OR$ then π_G will contain the clauses:

$$g_i(\mathbf{X}, \mathbf{Y}) \leftarrow g_{b_i}(\mathbf{X}, \mathbf{Y}); \qquad g_i(\mathbf{X}, \mathbf{Y}) \leftarrow g_{c_i}(\mathbf{X}, \mathbf{Y}).$$

If $a_i = NOT$ then π_G will contain the clause:

$$g_i(\mathbf{X},\mathbf{Y}) \leftarrow \overline{g_{b_i}}(\mathbf{X},\mathbf{Y}).$$

If $a_i = IN$ and g_i is the jth input of the circuit according to the order implied by the $2n$-tuple $(\mathbf{X},\mathbf{Y})$, then π_G will contain the clause:

$$g_i(Z_1,\ldots,Z_{j-1},1,Z_{j+1},\ldots,Z_{2n}) \leftarrow d$$

where d is a propositional letter. Moreover, for each i, $1 \leqslant i \leqslant k$, π_G contains the following three clauses:

$$\overline{\overline{g_i}}(\mathbf{X},\mathbf{Y}) \leftarrow g_i(\mathbf{X},\mathbf{Y}); \qquad \overline{\overline{g_i}}(\mathbf{X},\mathbf{Y}) \leftarrow \overline{g_i}(\mathbf{X},\mathbf{Y})$$

$$non\_3\_col \leftarrow g_i(\mathbf{X},\mathbf{Y}), \overline{g_i}(\mathbf{X},\mathbf{Y}).$$

Now let π'_{co3COL} be the following program:

$$non\_3\_col \leftarrow g_k(\mathbf{X},\mathbf{Y}), blue(\mathbf{X}), blue(\mathbf{Y})$$

$$non\_3\_col \leftarrow g_k(\mathbf{X},\mathbf{Y}), red(\mathbf{X}), red(\mathbf{Y})$$

$$non\_3\_col \leftarrow g_k(\mathbf{X},\mathbf{Y}), green(\mathbf{X}), green(\mathbf{Y})$$

$$hascolor(\mathbf{X}) \leftarrow blue(\mathbf{X})$$

$$hascolor(\mathbf{X}) \leftarrow red(\mathbf{X})$$

$$hascolor(\mathbf{X}) \leftarrow green(\mathbf{X})$$

where $blue(\mathbf{X})$ means that node identified by n-tuple $\mathbf{X}$ is colored in blue, and analogously for predicates *red*, *green*. $hascolor(\mathbf{X})$ means that the node has been assigned at least one color. Let $\pi_{cc} = \pi_G \cup \pi'_{co3COL}$. Let the input database D store only one ground atomic formula, namely, d. Moreover, let the query γ be defined as the following restricted clause:

$$((\forall \mathbf{X},\mathbf{Y})\overline{\overline{g_1}}(\mathbf{X},\mathbf{Y}) \wedge \cdots \wedge \overline{\overline{g_k}}(\mathbf{X},\mathbf{Y}) \wedge hascolor(\mathbf{X})) \rightarrow non\_3\_col.$$

Let $P = \{g_1,\ldots,g_k,\overline{\overline{g_1}},\ldots,\overline{\overline{g_k}},hascolor,non\_3\_col,d\}$ and $Q = \{\overline{g_1},\ldots,\overline{g_k}\}$.

We are then in the position to state the following result regarding the combined complexity of NRDATALOG^{CIRC}.

Theorem 5.2. *Under combined complexity, the problem of determining, given a* NRDATALOG^{CIRC} *program* $\langle \pi; P; Q; \emptyset \rangle$, *a database* D *and a restricted clause* γ, *if* π, $D \models_{P;Q;\emptyset} \gamma$ *is hard for co-NE.*

Proof. Consider the construction above. We next show that $\pi_{cc}, D \not\models_{P;Q;\emptyset} \gamma$ if and only if the graph represented by the circuit is 3-colorable.

Only if part. Let us assume that $\pi_{cc}, D \not\models_{P;Q;\emptyset} \gamma$; then there is a $(P;Q;\emptyset)$-minimal model M of $\pi_{cc} \wedge D$ which is not a model of γ, i.e., (*i*) $M \models (\forall \mathbf{X},\mathbf{Y})(\overline{\overline{g_1}}(\mathbf{X},\mathbf{Y}),\ldots,$

$\overline{\overline{g_k}}(\mathbf{X},\mathbf{Y})$) and $M \models (\forall \mathbf{X})(hascolor(\mathbf{X}))$ and (*ii*) $M \not\models non\_3\_col$. Since *hascolor* is a minimized predicate, it follows from (*i*) above that $M \models (\forall \mathbf{X})(red(\mathbf{X}) \vee blue(\mathbf{X}) \vee green(\mathbf{X}))$, i.e., each node has at least one color. Similarly, for all i, $(1 \leqslant i \leqslant k)$, as $M \models (\forall \mathbf{X},\mathbf{Y})\overline{\overline{g_i}}(\mathbf{X},\mathbf{Y})$ and since for all i, $(1 \leqslant i \leqslant k)$, $\overline{\overline{g_i}}$ is a minimized predicate, it follows from (*i*) above that for all i, $(1 \leqslant i \leqslant k)$, $M \models (\forall \mathbf{X},\mathbf{Y})(g_i(\mathbf{X},\mathbf{Y}) \vee \overline{g_i}(\mathbf{X},\mathbf{Y}))$. We next show that:

(α) for each $2n$-tuple $(\mathbf{e},\mathbf{d})$, $M \models g_i(\mathbf{e},\mathbf{d})$ if and only if the gate $g_i = (a_i, b_i, c_i)$ outputs 1 when the $2n$-tuple $(\mathbf{e},\mathbf{d})$ is presented as the input to the circuit; in particular, $M \models g_k(\mathbf{e},\mathbf{d})$ if and only if the nodes $\mathbf{e}$ and $\mathbf{d}$ are connected by an edge.

Indeed, if (α) turns out to be true, then it would follow from the rules above that (iii) all nodes are colored, and, (iv) since $M \not\models non\_3\_col$ the nodes connected by an edge are not colored using the same color. In turn, (iii) and (iv) would immediately imply that the input graph is 3-colorable.

Proof. (α). First of all note that, since $M \models (\forall \mathbf{X},\mathbf{Y})(g_i(\mathbf{X},\mathbf{Y}) \vee \overline{g_i}(\mathbf{X},\mathbf{Y}))$ and $M \not\models non\_3\_col$, it follows that (*v*) $M \models g_i(\mathbf{e},\mathbf{d})$ if and only if $M \not\models \overline{g_i}(\mathbf{e},\mathbf{d})$. We now proceed by induction on i.

Base, $i = 1$. Clearly, $a_1 = IN$. Therefore, π_G contains the rule

$$g_1(Z_1,\ldots,Z_{j-1},1,Z_{j+1},\ldots,Z_n) \leftarrow d$$

where j is the position of the input gate g_1 according to the arrangements of the inputs in the $2n$-tuple given as the input to the circuit. Since M is a model for $\pi_{cc} \wedge D$, it follows that $M \models d$. Therefore, since g_i is a minimized predicate, we have: $M \models g_1(\mathbf{e},\mathbf{d})$ if and only if the jth bit of $(\mathbf{e},\mathbf{d})$ is set to 1 if and only if the gate g_1 outputs 1 when the $2n$-tuple $(\mathbf{e},\mathbf{d})$ is presented as the input to the circuit.

Induction. Assume the statement holds for each $j < i$. We proceed by case analysis.

Case 1: $a_i = IN$. The same reasoning as above applies.

Case 2: $a_i = AND$. In this case, π_{cc} contains the rule:

$$g_i(\mathbf{X},\mathbf{Y}) \leftarrow g_{b_i}(\mathbf{X},\mathbf{Y}), g_{c_i}(\mathbf{X},\mathbf{Y}).$$

Then we have: $M \models g_i(\mathbf{e},\mathbf{d})$ if and only if (as g_i is minimized) $M \models g_{b_i}(\mathbf{e},\mathbf{d})$ and $M \models g_{c_i}(\mathbf{e},\mathbf{d})$ if and only if (by induction, as both b_i and c_i are less than i) both g_{b_i} and g_{c_i} outputs 1 when the $2n$-tuple $(\mathbf{e},\mathbf{d})$ is presented to the circuit as the input if and only if g_i outputs 1 when the $2n$-tuple $(\mathbf{e},\mathbf{d})$ is presented to the circuit as the input.

Case 3: $a_i = OR$. In this case, π_{cc} contains the rules:

$$g_i(\mathbf{X},\mathbf{Y}) \leftarrow g_{b_i}(\mathbf{X},\mathbf{Y}), \qquad g_i(\mathbf{X},\mathbf{Y}) \leftarrow g_{c_i}(\mathbf{X},\mathbf{Y}).$$

Then we have: $M \models g_i(\mathbf{e},\mathbf{d})$ if and only if (as g_i is minimized) $M \models g_{b_i}(\mathbf{e},\mathbf{d})$ or $M \models g_{c_i}(\mathbf{e},\mathbf{d})$ if and only if (by induction, as both b_i and c_i are less than i) either g_{b_i} or g_{c_i} outputs 1 when the $2n$-tuple $(\mathbf{e},\mathbf{d})$ is presented to the circuit as the input if and only if g_i outputs 1 when the $2n$-tuple $(\mathbf{e},\mathbf{d})$ is presented to the circuit as the input.

Case 4: $a_i = NOT$. In this case, π_{cc} contains the rule:

$$g_i(\mathbf{X}, \mathbf{Y}) \leftarrow \overline{g_{b_i}}(\mathbf{X}, \mathbf{Y}).$$

Then we have: $M \models g_i(\mathbf{e}, \mathbf{d})$ if and only if (as g_i is minimized) $M \models \overline{g_{b_i}}(\mathbf{e}, \mathbf{d})$ if and only if (because of (v) above) $M \not\models g_{b_i}(\mathbf{e}, \mathbf{d})$ if and only if (by induction, as b_i is less than i) g_{b_i} outputs 0 when the $2n$-tuple $(\mathbf{e}, \mathbf{d})$ is presented to the circuit as the input if and only if g_i outputs 1 when the $2n$-tuple $(\mathbf{e}, \mathbf{d})$ is presented to the circuit as the input.

If part. Let us assume that the graph G encoded in the circuit is 3-colorable and let C be a 3-coloring. Let M be an Herbrand interpretation built as follows:

- for each pair of n-bit strings $\mathbf{e}$ and $\mathbf{d}$:
  - for each i $(1 \leqslant i \leqslant k)$, $M \models g_i(\mathbf{e}, \mathbf{d})$ if and only if the gate g_i of the circuit outputs 1 when the $2n$-tuple $(\mathbf{e}, \mathbf{d})$ is presented as the input to the circuit; in particular, $M \models g_k(\mathbf{e}, \mathbf{d})$ if and only if there is an edge in G between nodes corresponding to $\mathbf{e}$ and $\mathbf{d}$;
  - $M \models red(\mathbf{e})$ if and only if node corresponding to $\mathbf{e}$ is colored in red by C;
  - $M \models blue(\mathbf{e})$ if and only if node corresponding to $\mathbf{e}$ is colored in blue by C;
  - $M \models green(\mathbf{e})$ if and only if node corresponding to $\mathbf{e}$ is colored in green by C;
  - $M \models hascolor(\mathbf{e})$;
  - for each i $(1 \leqslant i \leqslant k)$, $M \models 0\overline{\overline{g_i}}(\mathbf{e}, \mathbf{d})$;
  - for each i $(1 \leqslant i \leqslant k)$, $M \models \overline{g_i}(\mathbf{e}, \mathbf{d})$ if and only if $M \not\models g_i(\mathbf{e}, \mathbf{d})$;
- $M \models d$
- $M \not\models non\_3\_col$.

Clearly M is a model of $\pi_{cc} \wedge D$ and is not a model of γ. All we have to prove is that M is $(P; Q; \emptyset)$-minimal. Let us assume that there is a model N of $\pi_{cc} \wedge D$ such that $N \leqslant_{(P;Q;\emptyset)} M$ and $M \not\leqslant_{(P;Q;\emptyset)} N$. This means that the extension of some predicate in P must decrease in N w.r.t. M. Since *non_3_col* is already false in M, and d cannot have its extension decreased (otherwise D would not be satisfied any longer), the only possibilities are to decrease either the extension of g_i or the extension of $\overline{\overline{g_i}}$, for some i $(1 \leqslant i \leqslant k)$, or the extension of *hascolor*. However, as the extension of one of the three predicates *red*, *blue* or *green* cannot decrease, as they are fixed, to decrease the extension of *hascolor* cannot take place. So, we consider the predicates g_i and $\overline{\overline{g_i}}$. As for each i $(1 \leqslant i \leqslant k)$, $\overline{g_i}$ is fixed, it cannot have its extension decreased in N w.r.t. M. Therefore it follows that the extension of $\overline{\overline{g_i}}$ can decrease in N w.r.t. M, for some i, if and only if the extension of g_i decreases for the same i. Next, we will show that the extension of g_i cannot decrease either. We proceed by induction on i.

Base, $i = 1$. Clearly, $a_i = IN$. The rule associated to g_i has its body (the atom d) satisfied in M. As the extension of d cannot be decreased in N w.r.t. M (see above), and since g_i is minimized, it follows that the extension of g_i cannot be decreased either, otherwise N would not be a model for $\pi_{cc} \wedge D$ (recall that the extension of g_i in M

consists of all $2n$-tuples $(\mathbf{e},\mathbf{d})$ that have their jth bit set to 1, where j is the position of the input gate g_i in the arrangements of inputs in $(\mathbf{e},\mathbf{d})$).

Induction. Assume that the extension of g_i cannot be decreased in N w.r.t. M for each $j<i$. We proceed by case analysis.

Case 1: $a_i = IN$. The same reasoning as before applies.

Case 2: $a_i = AND$. As g_i is minimized, it follows from the rules for g_i in π_{cc} that g_i can be decreased in N w.r.t. M only if either g_{b_i} or g_{c_i} can be decreased. However, both b_i and c_i are less than i. Therefore, by induction, it is possible to decrease neither the extension of g_{b_i} nor the extension of g_{c_i}. Therefore the extension of g_i cannot be decreased as well.

Case 3: $a_i = OR$. Analogous to Case 2.

Case 4: $a_i = NOT$. As g_i is minimized, it follows from the rules for g_i in π_{cc} that g_i can be decreased in N w.r.t. M only if $\overline{g_{b_i}}$ can be decreased. But $\overline{g_{b_i}}$ is a fixed predicate, and therefore cannot be decreased.

Therefore, such a model N does not exist, and M is $(P;Q;\emptyset)$-minimal.

This concludes the proof. □

Since DATALOG^{CIRC} is more general than NRDATALOG^{CIRC}, we immediately obtain the same hardness result for DATALOG^{CIRC}.

The above results can be strengthened if we restrict our attention to databases with at most two constant symbols. In such a case we can guess the extension of a predicate of arity n in non-deterministic time 2^n. Therefore the combined complexity of determining if a restricted clause follows from a given DATALOG^{CIRC} program $\langle \pi;P;Q;\emptyset\rangle$ and database D is in co-NE. The same argument applies to Herbrand Universes whose cardinality is bounded by a constant. Therefore we can state the following corollary.

Corollary 5.3. *Let k be a fixed constant. Under combined complexity, the problem of determining, given a* NRDATALOG^{CIRC} *(resp.,* DATALOG^{CIRC}*) program $\langle \pi;P;Q;\emptyset\rangle$, a database D whose domain has cardinality less than k and a restricted clause γ, if $\pi, D \models_{P;Q;\emptyset} \gamma$ is complete for co-*NE.

Similarly to the results presented in previous sections, we can simplify the form of the query to be a simple ground literal by using extra rules and varying predicates, as we do in the following. The NRDATALOG^{CIRC} program $\langle \pi^Z_{cc};P';Q;Z\rangle$ contains all the rules of $\langle \pi_{cc};P;Q;\emptyset\rangle$, plus the rules:

$$has\_color(X) \leftarrow z.$$

$$\overline{\overline{g_1}}(X,Y) \leftarrow z.$$

$$\cdots\cdots$$

$$\overline{\overline{g_k}}(X,Y) \leftarrow z.$$

$$p \leftarrow z, non\_3\_col.$$

where $P' = P \cup \{p\}$, and $Z = \{z\}$. The input database is the same as before, while the query γ' is defined by the literal:

$$\neg z.$$

The following corollary is easy to prove, using the same argument shown in Example 3.2.

Corollary 5.4. *Under combined complexity, the problem of determining, given a* NRDATALOG^{CIRC} *(resp.,* DATALOG^{CIRC}*) program* $\langle \pi; P; Q; Z \rangle$*, a database D whose domain has cardinality less than k and a literal l, if* $\pi, D \models_{P;Q;Z} l$ *is hard for co-*NE.

*Moreover, if the cardinality of the input database D is bound by a given constant k then the problem above is in co-*NE.

6. Discussion and conclusions

This paper defines DATALOG^{CIRC}, the language obtained from ascribing semantics to DATALOG programs using general circumscription and demonstrates its use in solving complex queries to relational databases by means of several examples.

We have studied both data and combined complexity and the expressive power of querying relational databases through DATALOG^{CIRC} programs: We have showed that querying through DATALOG^{CIRC} programs is co-NP-complete under the data complexity measure and hard for co-NE under the combined complexity measure; furthermore, the main result presented in this paper proves that *all* queries in DB-co-NP can be expressed using DATALOG^{CIRC} programs.

It is interesting to know whether we can isolate polynomial sub-cases for data complexity by means of some syntactic restrictions on DATALOG^{CIRC} programs. Two such restrictions are now briefly mentioned. In both cases, polynomiality is shown first by "projecting" the DATALOG^{CIRC} program on a suitable propositional formula. The formula is the conjunction of all possible ground instances of the rules of the program (clearly, a sufficient number of propositional letters must be used, minimized predicates generate letters in P, and so on). The rationale for such a translation is that the size of the propositional formula is polynomial in the size of the input database. As a consequence, if we end up with a class of propositional formulae for which circumscriptive inference is polynomial, then we have proven a polynomial case for data complexity in DATALOG^{CIRC}.

Example 3.1 showed that data complexity is co-NP-complete if there are no varying predicates and the maximum arity of fixed predicates is 1. If we "project" a DATALOG^{CIRC} program without varying predicates and whose fixed predicates have arity 0, we obtain a propositional formula and a partition $\langle P; Q; \emptyset \rangle$ of its alphabet such that the size of Q is bounded by a constant, i.e., exactly as many as the fixed predicates in the original formula. It is possible to show that in such a case inference

in propositional circumscription is polynomial, which implies that data complexity is polynomial.

Data complexity is also polynomial if there is at most one literal in the body of each rule, even if there are varying predicates. This can be proven by taking advantage of the complexity analysis of inference in propositional circumscription that is performed in [7].

Efficient implementation of DATALOG^{CIRC} queries (is it possible to design *ad hoc* optimization techniques to speed up evaluation of DATALOG^{CIRC} queries?) will be the subject of future research.

Acknowledgements

Thanks to Thomas Eiter for his precious comments on the drafts of this paper. Anonymous referees' comments and suggestions led to substantial improvements in the quality of this paper. The work of the first author has been supported by ASI (Italian Space Agency), MURST (Italian Ministry for University and Scientific and Technological Research) and CNR (Italian Research Council) under the SARI project. The work of the second author has been supported by EC under the EC-NSF joint project "Deus ex machina: non-determinism for deductive databases" and by MURST under the project "Sistemi formali e strumenti per basi di dati evolute".

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, Reading, MA, 1995.
[2] S. Abiteboul, V. Vianu, Expressive power of query languages, in: J.D. Ullman (Ed.), Theoretical Studies in Computer Science, Academic Press, New York, 1992.
[3] A. Aho, J.D. Ullman, Universality of data retrieval languages, in: Proc. 6th ACM Symp. on Principles of Programming Languages, ACM Press, New York, Addison-Wesley, Reading, MA, 1979, pp. 110–117.
[4] P. Bonatti, T. Eiter, Querying disjunctive databases through non-monotonic logics, Theoret. Comput. Sci. 160 (1996) 321–363.
[5] M. Cadoli, The complexity of model checking for circumscriptive formulae, Inform. Process. Lett. 44 (1992) 113–118.
[6] M. Cadoli, T. Eiter, G. Gottlob, Default logic as a query language, in: Proc. 4th Internat. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94), 1994, pp. 99–108, Extended version to appear in IEEE Trans. Knowledge Data Eng.
[7] M. Cadoli, M. Lenzerini, The complexity of propositional closed world reasoning and circumscription, J. Comput. System Sci. 48 (1994) 255–310.
[8] A. Chandra, D. Harel, Computable queries for relational databases, J. Comput. System Sci. 21 (1980) 156–178.
[9] A. Chandra, D. Harel, Structure and complexity of relational queries, J. Comput. System Sci. 25 (1982) 99–128.
[10] A. Chandra, D. Harel, Horn clause queries and generalizations, J. Logic Programming 1 (1985) 1–15.
[11] T. Eiter, G. Gottlob, Propositional circumscription and extended closed world reasoning are Π_2^p-complete, Theoret. Comput. Sci. 114 (1993) 231–245.

[12] T. Eiter, G. Gottlob, H. Mannila, Adding disjunction to datalog, in: Proc. 13th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS-94), 1994, pp. 267–278; Extended version to appear on ACM Trans. Database Systems.
[13] T. Eiter, N. Leone, D. Saccà, The expressive power of partial models in disjunctive deductive databases, in: Logic in Databases, Lecture Notes in Computer Science, vol. 1154, Springer, New York, 1996, pp. 245–264.
[14] R. Fagin, Generalized first-order spectra and polynomial-time recognizable sets, in: R.M. Karp (Ed.), Complexity of Computation, AMS, New York, 1974, pp. 43–74.
[15] M.R. Garey, D.S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, Freeman, San Francisco, CA, 1979.
[16] M. Gelfond, V. Lifschitz, Compiling circumscriptive theories into logic programs: Preliminary report, in: Proc 7th National Conf. on Artificial Intelligence (AAAI-88), 1988, pp. 455–459.
[17] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proc. 5th Logic Programming Symp., The MIT Press, Cambridge, MA, 1988, pp. 1070–1080.
[18] M. Gelfond, H. Przymusinska, Negation as failure: careful closure procedure, Artif. Intell. J. 30 (1986) 273–287.
[19] M. Gelfond, H. Przymusinska, T. Przymusinsky, On the relationship between circumscription and negation as failure, Artif. Intell. J. 38 (1989) 49–73.
[20] Y. Gurevich, S. Shelah, Fixed-point extension of first-order logic, Ann. Pure Appl. Logics 32 (1986) 265–280.
[21] N. Immerman, Relational queries computable in polynomial time, Inform. and Control 68 (1986) 86–104.
[22] P. Kanellakis, Elements of relational database theory, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. B, Elsevier, Amsterdam, 1990, ch. 17.
[23] P.G. Kolaitis, C.H. Papadimitriou, Why not negation by fixpoint?, J. Comput. System Sci. 43 (1991) 125–144.
[24] V. Lifschitz, Computing circumscription, in: Proc. 9th Internat. Joint Conf. on Artificial Intelligence (IJCAI-85), 1985, pp. 121–127.
[25] V. Lifschitz, On the declarative semantics of logic programs with negation, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, Los Altos, 1988, pp. 177–192.
[26] J.F. Lynch, Complexity classes and theories of finite models, Math. Systems Theory 15 (1982) 127–144.
[27] J. McCarthy, Circumscription – a form of non-monotonic reasoning, Artif. Intell. J. 13 (1980) 27–39.
[28] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, Reading, MA, 1994.
[29] T. Przymusinski, On the declarative semantics of stratified deductive databases and logic programs, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, Los Altos, 1988, pp. 193–216.
[30] R. Reiter, Circumscription implies predicate completion (sometimes), in: Proc. 2nd National Conf. on Artificial Intelligence (AAAI-82), 1982, pp. 418–420.
[31] D. Saccà, Multiple stable models are definitely needed to solve unique solution problems, Inform. Process. Lett. 58 (1996) 249–254.
[32] Y. Sagiv, Optimizing datalog programs, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, Los Altos, 1988, pp. 659–698.
[33] C. Sakama, K. Inoue, Embedding circumscriptive theories in general disjunctive programs, in: Proc. 3rd Internat. Workshop on Logic Programming and Non-monotonic Reasoning (LPNMR-95), Lecture Notes in Computer Science, vol. 928, Springer, New York, 1995, pp. 344–357.
[34] J.S. Schlipf, Decidability and definability with circumscription, Ann. Pure Appl. Logic. 35 (1987) 173–191.
[35] J.S. Schlipf, The expressive powers of the logic programming semantics, J. Comput. System Sci. 51 (1995) 64–86.
[36] J.S. Schlipf, A survey of complexity and undecidability results for logic programming, Ann. Math. Artif. Intell. 15 (1995) 257–288.
[37] I.Stewart, Comparing the expressibility of languages formed using NP-complete operators, J. Logic Comput. 1 (3) (1991) 305–330.
[38] J.D. Ullman, Principles of Database and Knowledge Base Systems, vol. 1, Computer Science Press, Rockville, MD, 1988.

[39] M.Y. Vardi, The complexity of relational query languages, in: Proc. 14th ACM SIGACT Symp. on Theory of Computing (STOC-82), 1982, pp. 137–146.

[40] M.H. van Emden, R.A. Kowalski, The semantics of predicate logic as a programming language, J. ACM. 23 (4) (1976) 733–742.

[41] A. van Gelder, The alternating fixpoint of logic programs with negation, in: Proc. 8th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS-89), 1989, pp. 1–10.