Complexity Tailored Design: A New Design Methodology for Databases With Incomplete Information*

Tomasz Imielinski
Department of Computer Science,
Rutgers University,
New Brunswick NJ 08903
imielins@aramis.rutgers.edu

Ron van der Meyden
Information Processing Principles Group,
NTT Basic Research Laboratories,
3-1 Wakamiya Morinosato,
Atsugi-Shi, Kanagawa 243-01, Japan.
meyden@ntt-20.ntt.jp

Kumar V. Vadaparty
Department of Computer Engineering and Science
Case Western Reserve University
Cleveland OH44106
kumarv@alpha.ces.cwru.edu

To appear in *Journal of Computer and Systems Sciences*

*Authors are listed in alphabetical order. Work begun while all the authors were at Rutgers university.

Proposed Running Head: Complexity Tailored Design.

Author to be contacted

Kumar Vadaparty,
Department of Computer Engineering and Science,
Case Western Reserve University,
Cleveland, Ohio 44106. email: kumarv@truth.ces.cwru.edu, Phone: (216) 368 6197.

Abstract

This paper introduces a new design methodology, *Complexity Tailored Design*, for databases containing incomplete information. The novelty of the approach is that it introduces query complexity as a consideration in database design. In general, the problem of evaluating queries in databases with incomplete information is intractable. Complexity Tailored Design consists in setting the values of database design parameters so as to guarantee the efficient (polynomial time) evaluability of selected queries. We develop this approach for *OR-databases*, which are obtained by extending relational databases with *OR-objects* that express restricted disjunctions; OR-objects occur naturally in planning, scheduling and design applications. We consider two design parameters: (i) *typing* that delimits the columns in which OR-objects may occur, and (ii) *degree of coreference* that determines how OR-objects may be repeated within the database. We establish, for a broad class of queries in OR-databases, a complete syntactic characterization that *classifies the data complexity of each query as being either in PTIME or co-NP complete*, given a setting of the database design parameters. This characterization can then be used in complexity tailored design to determine "maximal" schemata for which the queries of interest to the user are evaluable in PTIME.

Contents

1 Introduction

This paper introduces *Complexity Tailored Design*, a new design methodology for databases containing incomplete information. The novelty of the approach is that it introduces query complexity as a consideration in database design. In general, queries in incomplete databases are intractable. Complexity Tailored Design consists in the setting of database design parameters so as to guarantee efficient (polynomial time) evaluability of the query, or set of queries, of interest. This requires a detailed analysis of the relationship between the complexity of particular queries and the settings of the design parameters. We develop such an analysis in this paper for one class of incomplete databases. Our analysis is comprehensive in that it decides the tractability or intractability of each query, with respect to each setting of the design parameters. This makes it possible to select settings of the design parameters that are optimal in the sense of allowing a maximal set of database instances, while maintaining efficient evaluability of the selected queries.

The particular class of incomplete databases we consider is the class of OR-databases. These databases extend relational databases by the addition of OR-objects, a type of domain-restricted null value expressing a limited form of disjunctive information. For example, a fact such as "*Part#1 can be implemented using Nickel or Cobalt*" can be encoded as $Implement(Part\#1, o)$ where o is an OR-object whose domain is the set $\{Nickel, Cobalt\}$. Thus, an OR-object represents a set of possible choices. An OR-database is a relational database whose tuples contain OR-objects as well as constants. Intuitively, an OR-database represents several possible worlds, or models, each obtained by replacing every OR-object by one of the choices it represents. Incomplete information of this sort arises in database applications involving planning, scheduling and design, as discussed in [5, 6]. Closely related to OR-objects are the "domain variables" of logic programs with finite constraints [13].

We say that a query (a closed formula of first order logic) is a consequence of an OR-database in case it holds in *all* models of the database. This is the standard query modality studied in the literature on logical databases[1] [15, 1, 11]. We will measure the complexity of query answering using the notion of *data complexity* [14, 2]. The data complexity of a *fixed* query Φ for a class $\mathcal{D}$ of databases is the complexity of determining, for any $D \in \mathcal{D}$, whether Φ is a consequence of the data D. The notion of data complexity is well motivated: for relational databases, it is known that the data complexity of first order queries is in PTIME, whereas if we measure query complexity as a function of the query size as well as the size of the database, we obtain a problem which is co-NP complete. By factoring out the size of the query, data complexity takes into account the fact that queries are usually small in comparison with the size of the database, which may grow to be very large as data accumulates over time. Data complexity also models situations in which we frequently wish to ask the same query as the database varies.

Unfortunately, the need to consider many possible worlds, instead of a single possible world, as

[1] However, other query modalities are also of interest in OR-databases. For example, one may wish to know whether a query is true in *some* model of the database [6].

in the case of relational databases, makes even simple queries intractable in OR-databases. For example, as we show in this paper, the data complexity of the query $\Phi \equiv \exists x[P(x) \wedge Q(x)]$ with respect to OR-databases is co-NP complete. Thus, the data complexity of even conjunctive queries jumps from PTIME to co-NP complete in moving from relational databases to OR-databases.

However, under certain restrictions on the occurrence of incomplete information in the database, the query Φ can be shown to be in PTIME. For example, if we restrict our attention to the class $\mathcal{D}'$ of OR-databases in which OR-objects appear only in the relation P, then we can show that the data complexity of Φ is in PTIME. This identifies a tractable class of databases for Φ. In other words, if the user requires that the query Φ be evaluable in PTIME, then the database administrator can disallow OR-objects in Q and thus force this query to be tractable.

In *complexity tailored design*, we identify database parameters responsible for query complexity. If the influence of these parameters on query complexity is understood, we may select settings of the parameters that will guarantee low complexity for the query, or set of queries, we wish to evaluate. These settings may then be incorporated into the database schema.

The design parameters for OR-databases we consider in this paper concern the occurrence of OR-objects in the database. They are the *typing function* and *degree of co-referencing*. A *typing function* specifies whether each column of each table is of (i) OR-type, in which case the column may contain both constants and OR-objects, or (ii) ATOMIC-type, in which case the column may contain constants *only*[2]. The *degree of co-referencing* of a database is a measure of the extent of repeated occurrences of the same OR-object. There are three "degrees": *type I*, which does not allow repetition of OR-objects at all, *type II*, which does not allow repetitions across columns and *type III*, which allows unrestricted repetitions.

Both *typing function* and *degree of co-referencing* restrict the information that can appear in the database instances in a natural way. A natural example of a typing constraint is *no OR-objects are allowed in the column named "social-security-number"*. An example requiring *type II* co-referencing is a fact such as "*Kumar and Ron take the same course which is either CS541 or CS540*" which is encoded by $TAKE(Kumar, o_1)$, $TAKE(Ron, o_1)$ where the domain of the OR-object o_1 is the set $\{CS540, CS541\}$. Clearly, these design parameters can be easily enforced, and they may be incrementally checked when the database is updated.

The following example illustrates how the data complexity of a particular query varies as a function of these design parameters, and how this information may be used to select settings that guarantee efficient evaluability of this query.

Example 1.1 (Complexity Tailored Design)
Consider the relations $P(C_1, C_2, C_3)$, $Q(C_4, C_5)$, $R(C_6, C_7)$, $S(C_8, C_9)$ and $T(C_{10})$ where the C_i are column names. The results of this paper enable us to determine the data

[2] Schema parameters closely related to this are enforced in existing systems. For example, the standard relational manipulation/query language SQL allows the database designers to state which columns can not take null values [10].

complexity of the query

$$\Phi \equiv \quad \exists xyzu\ [P(x,x,y) \wedge Q(x,y) \wedge R(y,z) \wedge S(z,u) \wedge T(u)]$$

for different degrees of co-referencing and for different typing functions (see the adjoining table.)

α_1: OR-objects are allowed in columns C_1, C_2, C_3, C_5, C_7 and C_9 only.
α_2: OR-objects are allowed in columns C_1, C_2, C_5, C_7 and C_9 only.
α_3: OR-objects are allowed in C_1, C_5, C_7 and C_9 only.
α_4: OR-objects are allowed in C_5, C_7 and C_9 only.
α_5: OR-objects are allowed in C_5,C_7 only.

Note that in each case, α_{i+1} is obtained from α_i by constraining one additional column to be ATOMIC.

Co-Referencing	α_1	α_2	α_3	α_4	α_5
type I	co-NP	PTIME	PTIME	PTIME	PTIME
type II	co-NP	co-NP	co-NP	PTIME	PTIME
type III	co-NP	co-NP	co-NP	co-NP	PTIME

An entry "co-NP" in this table indicates that the query has co-NP complete data complexity with respect to the class of databases satisfying the given parameter settings; an entry "PTIME" indicates that the query is polynomial time evaluable with respect to those settings.

The results of this analysis may be used to determine restrictions on the database schema. Suppose it is essential to the application that the query Φ be evaluable in polynomial time. The table above shows that this complexity may be attained with respect to the class $\mathcal{D}(\alpha_4, II)$ of databases specified by typing function α_4 and degree of co-reference type II, i.e., multiple occurrences of OR-objects constrained to lie within the same column. Therefore, in order to ensure polynomial time evaluation of the query Φ, we may decide as a part of the database design process to restrict database instances to those conforming to typing function α_4 and degree of co-referencing type II. Alternately, the query also has polynomial time data complexity in the class $\mathcal{D}(\alpha_2, I)$ of databases specified by typing function α_2 and degree of co-reference type I. Since these classes of databases are orthogonal (neither is contained in the other) this constitutes another candidate for restriction of database instances.

Since setting database design parameters as illustrated above involves restricting the set of possible database instances, it is desirable to guarantee that the settings we choose are optimal. That is, we would like to have a maximal class of allowable database instances (within the vocabulary of the design parameters) which will ensure that the queries of interest have PTIME data complexity.

Since queries in general have co-NP complete data complexity in OR-databases, and since we do not know for certain that co-NP$\neq$P, we cannot expect to be able to provide such guarantees. However, it is possible to guarantee optimality in a weaker sense: for those settings of the parameters for which we do not know data complexity to be in PTIME, we can show co-NP completeness. Under the assumption that co-NP$\neq$P, this is the best possible. The following example illustrates the selection of parameter settings optimal in this sense.

Example 1.2 (Optimal Parameter Settings)
In the above example, note that if we attempt to generalize the class of databases $\mathcal{D}(\alpha_4, II)$ to the larger class $\mathcal{D}(\alpha_3, II)$ by allowing OR-objects to occur also in the column C_1, then the data complexity of the query Φ jumps to co-NP complete. It can in fact be shown using the results of the paper that the same jump in complexity occurs if any other additional column is permitted to contain OR-objects. Thus, the class given by (α_4, II) is, within the vocabulary of the design parameters, a maximal class for which we are able to ensure PTIME data complexity of the query Φ. (On the other hand, note that if we restrict the degree of co-reference to be type I, then we may add two further OR-columns while remaining in PTIME. In addition, as the table shows, attempting to enlarge the class $\mathcal{D}(\alpha_2, I)$ by liberalizing the degree of co-reference to type II also results in a jump in data complexity.)

In general, there is not a unique maximal setting for the design parameters which ensures PTIME data complexity. It can be shown that (α_5, III) also specifies a maximal schema, as does (α_2, I). This illustrates that there is a tradeoff between the set of columns containing OR-objects and the degree of co-referencing. Furthermore, there can be many maximal typing functions with respect to a fixed co-reference class which ensure PTIME data complexity of the query. For example, the schema in which we permit columns C_6, C_8 and C_{10} to contain OR-objects with type II co-referencing may also be shown to be maximal while ensuring PTIME data complexity.

In order to perform complexity tailored design as illustrated above, it is necessary to have a detailed analysis of the relationship between the data complexity of the particular queries of interest and various design parameters. This requires results considerably stronger than those usually established in the literature on incomplete databases (e.g. [1, 15]). Generally, results in this area state that some *class* of queries $\mathcal{Q}$ has data complexity characterized by a complexity class $\mathcal{C}$ with respect to some class of databases $\mathcal{D}$, in the sense that

(i) all queries in $\mathcal{Q}$ have data complexity in class $\mathcal{C}$ with respect to databases $\mathcal{D}$, and

(ii) there exist a query in $\mathcal{Q}$ that has data complexity complete in class $\mathcal{C}$ for databases $\mathcal{D}$.

Such a result provides a tight upper bound on the data complexity of *all* queries in $\mathcal{Q}$. However, except for the query used to establish (ii), this leaves open the precise data complexity of *particular* queries, which is what we need in order to carry out a complexity tailored design. Given a query Φ and a class $\mathcal{D}$ consisting of all databases conforming to a given schema, we need to be able to determine if the data complexity of Φ for $\mathcal{D}$ is in PTIME or intractable (co-NP complete).

The main technical contribution of this paper is to develop this stronger form of result for the class of *proper queries*: these are the existentially quantified conjunctive queries in which no two atoms have the same predicate symbol. The body of the paper consists of the analysis necessary to completely characterize the complexity of proper queries in OR-object databases with respect to the schema parameters introduced above[3]. We structure this characterization as follows.

We observe that the complexity of processing a query in a class of databases $\mathcal{D}$ depends on (i) the pattern of variables in the query (ii) the typing function α and (iii) the degree of co-referencing d to which $\mathcal{D}$ conforms. Thus, for each degree of co-referencing, the data complexity of Φ in $\mathcal{D}$ depends on the pair $\langle \alpha, \Phi \rangle$. We call such pairs ornated queries. We define certain syntactic properties of ornated queries, namely *strongly cyclicity*, *cyclicity* and *tri-connectedness*. These properties yield a complete syntactic characterization of data complexity which is summarized in Figure 1.

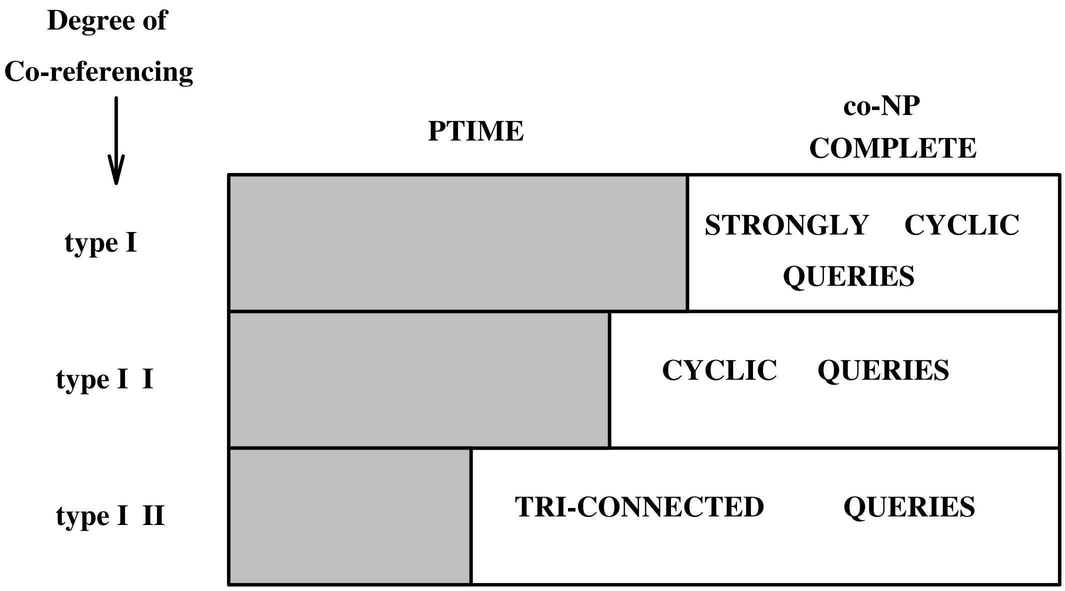

Figure 1: A complete syntactic characterization of proper queries

The results of Figure 1 are to be interpreted as follows: Consider a query Φ, a typing function α and let the degree of co-reference be of *type I*. Let $\mathcal{D}$ denote the class of *type I*-databases that conform to α. Then the Figure 1 states that the complexity of processing Φ in $\mathcal{D}$ is co-NP complete

[3]The results of this paper completely subsume the results of Imielinski and Vadaparty [7]. In [7] Imielinski and Vadaparty stated a complete characterization which is correct for proper queries in type II databases. However this classification does not apply to type I databases or to type III databases, nor does it extend to improper queries, as was erroneously claimed. In this paper we focus on proper queries and show how to extend the classification of [7] to type I and type III classes of databases.

if $\langle \Phi, \alpha \rangle$ is strongly cyclic and is PTIME otherwise. The other two cases for *type II* and *type III* can be interpreted analogously. The figure also conveys the containment relationships between the various classes of ornated queries. For example, the set of ornated queries in PTIME with respect to type III databases is a proper subset of the set of ornated queries in PTIME with respect to type II databases.

The structure of the paper is as follows. Section 2 gives some basic definitions regarding OR-objects, databases with OR-objects, and data complexity of a query. The rest of the paper can be divided into two main parts.

The first part identifies proper queries whose data complexity is in PTIME. This classification depends on a property called the "representative model property", which is similar to the "definite answer property" well-known from logic programming theory [9]. This property is discussed in Section 3. Using this property, Section 4, Section 5 and Section 6 identify queries whose data complexity is in PTIME for type I, type II and type III databases, respectively.

The second part identifies proper queries whose data complexity is co-NP complete. The tools needed to establish this classification are presented in Section 7. Using these tools, Section 9, Section 8 and Section 10 identify the queries whose complexity is co-NP complete for type I, type II and type III databases, respectively.

For *every* degree of co-referencing, all queries that are not shown to be in PTIME in the first part are shown to have co-NP complete data complexity in the second part. Thus every proper conjunctive query is shown to have either PTIME data complexity or co-NP complete data complexity. Thus we achieve a *complete syntactic characterization.*

Section 11 contains further discussion and concluding remarks. In Appendix C we briefly consider queries that are not proper, and give some examples that illustrate how the results of the paper break down under this generalization. While we do not know how to classify of the complexity of queries in this class, we do provide some sufficient conditions and some necessary conditions for such queries to have various complexities.

2 Basic Definitions

This section introduces the basic vocabulary to define OR-databases, schemas, models and entailment. A relation in our formalism is denoted by $R(C_1, \ldots, C_k)$ where R is a relation-name taken from a denumerable set $\mathcal{R}$ and the C_i's are column-names taken from a denumerable set $\mathcal{C}$. This notation is similar to the notation for relations in the relational model [12]. However, unlike in the relational model, where relations may contain only constants, a relation in our formalism may contain OR-objects as well as constants. The presence of OR-objects in the relations introduces two more parameters in the schema specification of our formalism: (i) a *typing function* that determines which columns of each relation can contain OR-objects, and (ii) *the degree of co-referencing* that determines what patterns of repetition of OR-objects are allowed.

Definition 2.1 (OR-schema)
A OR-database schema S, called an OR-schema, is a tuple of the form

$$S = \langle \{R_1(C_{11}, \ldots, C_{1a_k}), \ldots, R_m(C_{m1}, \ldots, C_{ma_m})\}, \quad \alpha, \quad \delta \rangle$$

where the R_i are relation-names taken from $\mathcal{R}$, the C_{ij} are distinct column-names taken from $\mathcal{C}$, α is a typing function that maps column-names to either A or O, and δ is a degree of co-referencing, which can be either I, II or III. A column C is called an OR-column if $\alpha(C) = O$ and it is called an ATOMIC-column if $\alpha(C) = A$.

An OR-database D consists of a finite set of finite relations over the set $\mathcal{U} \cup \mathcal{V}$, where $\mathcal{U}$ is a set of constants and $\mathcal{V}$ is a set of *OR-objects*, together with a *domain mapping Dom* which assigns to each OR-object $o \in \mathcal{V}$ occurring in D a non-empty finite set of constants from $\mathcal{U}$. Equivalently, we will view OR-databases as consisting of a domain function *Dom* together with a finite set of atoms of the form $P(u_1, \ldots, u_n)$ where P is some predicate symbol and the u_i are either constants or OR-objects.

We will consider three classes of OR-databases, corresponding to different assumptions about the repetition of OR-objects. An OR-database D is a *type I* database if each OR-object o occurs precisely once in D. An OR-database D is a *type II* database if no OR-object occurs in more than one column. The class of *type III* OR-databases places no constraints on the occurrence of OR-objects. In type III databases OR-objects may be repeated both within and across different columns. Note that every type I database is a type II database, and every type II database is a type III database.

An OR-database is an *instance* of an OR-schema if it *conforms* to the schema, where conformance is defined as follows:

Definition 2.2 (An instance of an OR-schema)
Let $S = \langle \{R_1(C_{11}, \ldots, C_{1a_k}), \ldots, R_m(C_{m1}, \ldots, C_{ma_m})\}, \quad \alpha, \quad \delta \rangle$ be an OR-schema and D an OR-database. Then D conforms to S if

- *The relations of D correspond in name and arity to the relations specified in S.*
- *If C is an ATOMIC-column, then the column of D corresponding to C contains only constants. If C is an OR-column, then OR-objects also may occur in this column of D.*
- *D is a type δ database.*

Example 2.1 (OR-Schema and OR-database)
Consider a schema $S = \langle\langle TEACH(Teacher, Course_1), TAKE(Student, Course_2)\rangle, \alpha, III\rangle$ where $\alpha(Course_1) = \alpha(Course_2) = O$ and $\alpha(Student) = \alpha(Teacher) = A$. The following is an instance of S. Note that the OR-object o_1 appears in two columns, and hence the instance D is not of type I or type II.

TEACH

Tname	*Cno*
Tomasz	o_1
Diane	o_2
Chris	*421*

TAKE

Student	*Course*
Ron	o_1
Kumar	o_1
Pamila	u

DomainValues

$Dom(o_1) = \{CS541, CS542\}$
$Dom(o_2) = \{CS509, CS539\}$
$Dom(u) = \{CS671, CS672\}$

Table 3: *An instance D of the OR-Schema S.*

Note that the OR-object o_1 with $Dom(o_1) = \{CS541, CS542\}$ is shared by three different atoms, $TAKES(Ron, o_1)$, $TAKES(Kumar, o_1)$ and $TEACH(Tomasz, o_1)$. These atoms encode the fact that "Ron and Kumar take the course taught by Tomasz, and this course is either $CS541$ or $CS542$". Thus there is a co-reference within $TAKE$ and another co-reference across $TAKE$ and $TEACH$.

Intuitively, an OR-database represents a set of relational databases, each of which is obtained by substituting for every OR-object a value from its domain. Each of the relational databases thus obtained is called a *model* of the original OR-database.

Example 2.2 (A model)
The following is a model of the database instance shown in the previous example.

TEACH

Tname	*Cno*
Tomasz	*CS541*
Diane	*CS509*
Chris	*CS421*

TAKE

Student	*Course*
Ron	*CS541*
Kumar	*CS541*
Pamila	*CS672*

Choose CS541 for o_1
Choose CS509 for o_2
Choose $CS672$ for u

Table 4: *A model M of the instance of the previous example.*

Note that all the OR-objects are replaced by a member from their respective domains, as shown above.

A mapping θ which assigns to each OR-object o in a database D a constant in $Dom(o)$ is called a *valuation* of D. Valuations are substitutions which may be applied to databases: we write $D\theta$ for the result of substituting $\theta(o)$ for each occurrence of an OR-object o in D. In the above example, if we let $\theta = [CS541/o_1,\ CS509/o_2,\ CS672/u]$, then the model M above is given by $M = D\theta$. Note that we require that all occurrences of an OR-object o be replaced by the same value from $Dom(o)$ in constructing a model. It is clear that if an OR-object o has a singleton domain $Dom(o) = \{c\}$, then the database obtained by replacing all occurrences of o by the value c has precisely the same set of models. Hence we will generally assume that the domains of OR-objects have cardinality at least two, unless otherwise noted in certain places for the convenience of the proofs.

The query language $\mathcal{L}$ is a first order language built from the following vocabulary: the constant symbols of this language are drawn from the universal set $\mathcal{U}$ of constants from which databases are formed, and the variables of this language are from a denumerable set $\mathcal{W}$. For every relation of arity k, a k-place predicate is present in $\mathcal{L}$; there are no function symbols in $\mathcal{L}$. Any formula of $\mathcal{L}$ can be viewed in a natural way as a query. We consider only a subset of the formulas of this language as queries in our formalism.

Definition 2.3 (Proper Queries and Entailment)
A proper query is an existentially quantified conjunction of atomic formulae, no two of which have the same predicate symbol. To distinguish the atomic formulae of a query from the atoms of a database, we refer to the former as literals. *Given a closed proper query Φ and an OR-database D, we say that D entails Φ, denoted by $D \models \Phi$, if every model of D satisfies Φ.*

Thus, $\Phi \equiv \exists x[P(x) \wedge Q(x)]$ is a proper query while $\Phi' \equiv \exists xy[P(x,y) \wedge P(y,x)]$ is not a proper query because the predicate P is used in two different literals.

The main result of this paper is a complete syntactic characterization of the complexity of closed proper queries. That is, this characterization applies only to "yes/no" proper queries. The restriction to closed queries is frequently made when dealing with databases containing disjunctive information, because otherwise we need to consider disjunctive answers. However, some of our proofs will involve dealing with queries that have free variables.

Data complexity of a query measures the complexity of answering a query, given a database instance. This notion, originally developed for queries on relational databases, has been investigated by a number of researchers [2, 8, 14]. Here we extend this notion to OR-databases and study the complexity of querying OR-databases.

Definition 2.4 (Data Complexity)
The data complexity of a query Φ with respect to a schema S is the complexity of determining whether a given OR-database D belongs to the set $\{\ D \mid D$ conforms to S and $D \models \Phi\}$

Notation: Since the data complexity of Φ with respect to S depends on the pattern of variables in the query Φ, and on the typing function α and degree of co-referencing δ of S, it is convenient to combine the query and typing function into a single notation by using predicates with ornation.

Thus, a relation $P(C_1, C_2)$ with $\alpha(C_1) = O$ and $\alpha(C_2) = A$ is denoted by $P^{O,A}$ in any query involving the relation P. When we refer to the data complexity of an ornate query Φ for a given degree of co-referencing, what we mean is the data complexity of Φ with respect to the schema implied by the super-scripts of Φ and the degree.

For example, when we speak of the complexity of the query

$$\exists xyz[P^{O,A}(x,y) \wedge Q^{A,O,A}(y,z,z)]$$

with respect to type I databases we mean the data complexity of the query $\exists xyz[P(x,y) \wedge Q(y,z,z)]$ with respect to the schema $\langle\{P(C_1,C_2), Q(C_3,C_4,C_5)\}, \alpha, I\rangle$ where $\alpha(C_1) = \alpha(C_4) = O$ and $\alpha(C_2) = \alpha(C_3) = \alpha(C_5) = A$.

We use a short-hand notation to refer to literals in our queries. First, since we consider only proper queries, we can refer to each literal uniquely using the corresponding predicate as a sub-script. Second, we are often interested only in the *set* of variables occurring in the literal, not their order. Thus, a literal $P^{O,A,A,A,O}(x,y,y,y,x)$ is represented as $l_P(x,y)$, which conveys the information that the literal with predicate P has variables x and y.

3 Tools for PTIME Proofs

In this section we introduce a semantic property called the *representative model property* that enables us to characterize queries with PTIME data complexity. Then we define two syntactic notions, *strong marking* and *marking*, to capture the representative model property for type I and type II databases, respectively. Finally, we compare the representative model property with the definite answer property from logic programming [9].

3.1 Representative Model

The fundamental property that underlies all the polynomial time classes of queries we discuss in this paper is the following.

Definition 3.1 (Representative Model Property)
Let $\mathcal{C}$ be a class of databases and $\Phi(\vec{x})$ a query with free variables $\vec{x}$. Then $\Phi(\vec{x})$ is said to have the representative model property for $\mathcal{C}$ if for every database $D \in \mathcal{C}$ there exists a model m of D such that

$$\{\ \vec{a} \mid D \models \Phi(\vec{a})\} = \{\ \vec{a} \mid m \models \Phi(\vec{a})\}.$$

A model m of a database D satisfying this equation is called a representative model of D with respect to $\Phi(\vec{x})$.

Note that if the query in the above definition has no free variables, then a representative model of a database $D \in \mathcal{C}$ will be either a counter-model to the query (when $D \not\models \Phi$) or any model of the database D (when $D \models \Phi$). In the former case, $\{ \vec{a} \mid D \models \Phi(\vec{a})\}$ is the empty set $\emptyset$, while in the latter case we take $\{ \vec{a} \mid D \models \Phi(\vec{a})\}$ to be the set $\{\Lambda\}$ containing just the null vector Λ.

In general, the representative model property does not hold. For example, the query $P(x)$ does not have the representative model property for the class of databases with schema $\langle\{P(C)\}, \alpha, I\rangle$ where $\alpha(C) = O$. For, the database $\{P(o)\}$ with $Dom(o) = \{a, b\}$ constitutes a counter-example. This is because one model of the database satisfies $P(a)$, while the other satisfies $P(b)$. However, the database entails neither $P(a)$ nor $P(b)$. Hence neither model can serve as representative model.

For an example of a query and a class of databases satisfying the representative model property, consider the query $\Phi(x) = \exists y[P^{A,O}(x, y)]$ and the class of type I databases conforming to the typing of $\Phi(x)$. Clearly, for any database D in this class, $D \models \Phi(a)$ if and only if a appears in the first column of some tuple in D. Hence, any model of D will serve as a representative model.

Note that in general, a query with free variables may have disjunctive answers in the context of databases with disjunctive information. For example, the query $P(x)$ has the disjunctive answer $a \vee b$ in the database $\{P(o)\}$ with $Dom(o) = \{a, b\}$. However, if a query has the representative model property, then it never yields disjunctive answers for any database. Formally, if $\Phi(x)$ has the representative model property for the class $\mathcal{C}$ of databases, then for any database $D \in \mathcal{C}$, if $D \models \Phi(a_1) \bigvee \ldots \bigvee \Phi(a_n)$ then $D \models \Phi(a_i)$ for some a_i. This is because of the following. Let $D \models \Phi(a_1) \bigvee \ldots \bigvee \Phi(a_n)$. Then $m_R \models \Phi(a_1) \bigvee \ldots \bigvee \Phi(a_n)$ where m_R is a representative model. But, since m_R is a definite database, it follows that $m_R \models \Phi(a_i)$ for some a_i. Since m_R is representative, $D \models \Phi(a_i)$. Thus, any disjunctive answer is subsumed by a definite answer.

If a query $\Phi(x)$ has the representative model property for a class $\mathcal{C}$ of databases and if for any $D \in \mathcal{C}$ a representative model of D can be computed in PTIME, then the output of $\Phi(x)$ on D can also be computed in PTIME. This is because the representative model is a definite database. This observation underlies the PTIME algorithms that we obtain in later sections. First, we describe a property called marking that helps us to provide sufficient conditions for a query to have the representative model property.

3.2 Marking

A syntactic property that apparently results in *failure* of the representative model property is as follows: if a free variable of a query occurs in an OR-argument then the query will not have the representative model property. This has been illustrated above for $P^O(x)$. It is also possible for this property to fail indirectly. Consider the query $\Phi(x) = \exists y[P^O(y) \wedge Q^{A,A}(y, x)]$. Here the "disjunction" in the variable y is transmitted to the variable x. For example, let D be the database containing the atoms $\{P(o), Q(a, b), Q(c, d)\}$ where o is an OR-object with domain $Dom(o) = \{a, c\}$. Then neither of the two models of D is a representative model because they satisfy either $\Phi(b)$ or

$\Phi(d)$. Neither of these formulae is entailed by the database. Thus, the disjunction is "transmitted" from y to x in $\Phi(x)$.

The following definition is intended to capture this method of "transmission of disjunction".

Definition 3.2 (Marking Path)
A marking path *from a literal l_P to a variable y in a query Φ is a sequence of variables $\langle x_0, x_1, \ldots, x_n\rangle$ such that $x_n = y$ and*

1. *the variable x_0 occurs in an OR-argument of l_P, and*
2. *if $n > 0$ then Φ contains a literal (possibly equal to l_P) which contains both x_1 and an occurrence of the variable x_0 distinct from the occurrence of x_0 used to satisfy condition 1, and*
3. *$x_i \neq x_0$ for $i = 1 \ldots n$, and*
4. *for all $i = 1 \ldots n-1$ there exists a literal in Φ containing both variables x_i and x_{i+1}.*

If such a marking path exists then we say that the variable y is marked *by l_P. A literal l_P is said to* mark *another literal l_Q if there exists a marking path from l_P to some variable of l_Q.*

Note that the definition of marking depends only on the pattern of the variables and on the ornation but not on which variables of the query are quantified. Also note that a literal cannot mark itself, although there may exist a marking path from a literal to some of the variables occurring in that literal.

Example 3.1 (Marking)
Consider the query

$$\exists xyzw[P^{A,O}(x,y) \wedge Q^{A,O}(y,z) \wedge R^{A,A}(z,w)].$$

In this query the variable y is marked by the literal l_P through the path $\langle y\rangle$ and the variable w is marked by the literal l_P through the path $\langle y, z, w\rangle$. But the variable x is *not* marked by the literal l_P, or any other literal. For example, the path $\langle y, x\rangle$ violates constraint 2: no literal contains both x and an occurrence of y distinct from the occurrence of y in an OR-argument of l_P. On the other hand, the path $\langle y, z, y, x\rangle$ violates constraint 3.

In the query $\exists xy[P^{A,O,A}(x,y,y)]$ the variable x is marked by the literal l_P, the marking path being $\langle y, x\rangle$. Here the variable y occurs in an OR-argument of l_P (the second argument), and the literal l_P contains a distinct occurrence of the variable y (in the third argument), as well as an occurrence of the variable x. Note that x would also marked in this query if the type of the third argument were OR.

We have suggested informally that disjunction is transmitted through marking paths. A reasonable conjecture is that this is the only way that disjunction is transmitted, so that if all the free variables of a query are unmarked then the query has the representative model property. We will establish a result in Section 5 that shows something like this conjecture to be true with respect to type II databases.[4] However, for type III and type I databases the situation is more complicated. In the case of type III databases, marking paths are not the only way in which disjunctions may be transmitted from OR-arguments to output variables (see Section 6). On the other hand, in type I databases, it is possible for the transmission of disjunction along marking paths to be blocked, and even queries with marked free variables may have the representative model property.

Example 3.2 (The Representative Model Property in Type I Databases)
Consider the query $\Phi(x) \equiv P^{A,O}(x,x)$. Clearly, for the class of type II databases, $\Phi(x)$ does not have the representative model property. For, consider $D = \{P(a,o), P(b,o)\}$ with $Dom(o) = \{a,b\}$. This database has two models, one of which satisfies $P(a,a)$ while the other satisfies $P(b,b)$. But neither of these atoms is entailed by the database. Hence neither model is representative.

However, $\Phi(x)$ does have the representative model property for type I databases. Suppose that D is a type I database. As previously noted, each OR-object may be assumed to have at least two members in its domain, otherwise it may be replaced by a constant. For each atom $P(a,o) \in D$, define $\theta(o) = b$ such that $b \in Dom(o)$ but $b \neq a$. Now, $D\theta \models \Phi(a)$ iff $P(a,a) \in D$ iff $D \models P(a,a)$. This shows that the query $\Phi(x)$ has the representative model property in spite of the fact that the variable x is marked in this query. A similar argument shows that the query $\Phi(y) = \exists x[P^{O,A}(x,y) \wedge Q^{A,A}(x,y)]$ has the representative model property, although the variable y is marked.

However, the query $\Phi(x) \equiv \exists y[P^{O}(y) \wedge Q^{A,A}(y,x)]$ still does not have the representative model property on x for type I databases. The database $D = \{P(o), Q(a,b), Q(c,d)\}$ with $Dom(o) = \{a,c\}$ that was used to show this above is a type I database.

Thus, only certain kinds of marking result in the failure of the representative model property in the case of type I databases. Our classification of the complexity of queries in type I databases will make use of the following modified notion of marking.

Definition 3.3 (Strong Marking)
A marking path $\langle x_0, \ldots, x_n\rangle$ from the literal l_P to the variable x_n is called a strong marking path *from l_P to x_n if (i) x_0 has unique occurrence in l_P and (ii) for all $i = 1 \ldots n$ the variable x_i does not occur in l_P. If such a marking path exists then we say that the variable x_n is strongly marked by l_P. A literal l_Q is strongly marked by a distinct literal l_P if l_P strongly marks some variable of l_Q.*

[4] The result we prove strengthens both the antecedent and the consequent of the conjecture, so leaves open its precise status.

A literal l_Q is strongly marked in Φ if there exists a distinct literal l_P of Φ which strongly marks some variable of l_Q.

Example 3.3 (Strong Marking)
The variables x and y are not strongly marked in the query $\exists xy[P^{O,O}(x,x) \wedge Q^{A,A}(x,y)]$ because the variable x occurs twice in the literal l_P, violating the first condition of the definition of strong marking. Thus the literal l_Q is not strongly marked in this query.

In the query $\exists xy[P^{O,A}(x,y) \wedge Q^{A,A}(x,y)]$ the variable x is strongly marked but y is not strongly marked. The literal l_P marks x via the path $\langle x \rangle$. The only marking path from l_P to y is $\langle x, y \rangle$, but this is not a strong marking path because it violates the second condition of the definition of strong marking. The literal l_Q is strongly marked by l_P in this query because l_P strongly marks the variable x through the path $\langle x \rangle$.

We now note some properties of marking and strong marking. First, neither is transitive as a relation on the literals of a query.

Observation 3.1 (Non-Transitivity of Marking)
In the query

$$\exists xyz[P^{A,O}(x,y) \wedge Q^{A,A,O}(y,z,z) \wedge R^{A}(x)]$$

the literal l_P marks l_Q via the path $\langle y \rangle$ and l_Q marks l_R via the path $\langle z, y, x \rangle$. But l_P does not mark l_R. Thus, marking is not transitive. Note however, that l_Q marks l_P.

To see that strong marking is not transitive, consider the query

$$\exists xyz[P^{O,A}(x,y) \wedge Q^{A,O}(x,z) \wedge R^{A,A}(z,y) \wedge S^{A}(y)].$$

Here we have that l_P strongly marks l_Q and l_Q strongly marks l_S, but l_P does not strongly mark l_S, because every marking path from l_P to l_S passes through the variable y, which occurs in l_P, violating condition 2 of the definition. Notice that l_Q strongly marks l_P in this example via the path $\langle z, y \rangle$.

The following lemma shows that marking is *almost* transitive.

Lemma 3.2 (Transitivity of Marking)
Let l_P, l_Q and l_R be distinct literals in a query Φ, such that l_P marks l_Q and l_Q marks l_R. Then either l_P marks l_R, or l_Q marks l_P. The same result holds with "marks" replaced by "strongly marks" throughout.

Proof: Suppose that $\langle x_0, \ldots, x_n \rangle$ is a (strong) marking path from l_P to l_Q, and that $\langle y_0, \ldots, y_m \rangle$ is a (strong) marking path from l_Q to l_R. There are two possibilities: either the variable x_0 occurs in the sequence $y_0, \ldots, y_m$ or it does not. In the first case l_Q (strongly) marks x_0, since any prefix of a (strong) marking path is a (strong) marking path. In the second case the sequence $\langle x_0, \ldots, x_n, y_0, \ldots, y_m \rangle$ is a marking path from l_P to l_R. It may still not be a strong marking path, even though both of the original sequences are strong marking paths, since it is possible that some variable y_i occurs in l_P. In this event, however, l_Q strongly marks l_P. ∎

Here we digress to observe an interesting relation between the representative model property and the *definite answer property* from logic programming [9]): a query of the form $\exists x \Phi(x)$ is a consequence of a definite Horn clause program just in case there exists some term a such that $\Phi(a)$ is a consequence of the program. The following lemma shows that the representative model property is at least as strong as the definite answer property.

Lemma 3.3 *If $\Phi(x)$ has the representative model property for a class of databases $\mathcal{C}$, then $\exists x \Phi(x)$ has definite answer property on x for $\mathcal{C}$.*

Proof: Suppose that $\Phi(x)$ has the representative model property for $\mathcal{C}$. Let $D \in \mathcal{C}$ have representative model m_R. Then by definition

$$\{ \ a \mid m_R \models \Phi(a) \} = \{ \ a \mid D \models \Phi(a) \} \qquad (1)$$

Now suppose that $D \models \exists x \Phi(x)$
$\Rightarrow \quad m_R \models \exists x \Phi(x) \quad$ (m_R is a model of D)
$\Rightarrow \quad m_R \models \Phi(a)$ for some $a \quad$ (m_R is a definite database)
$\Rightarrow \quad D \models \Phi(a)$ by (1) ∎

However, the converse is not true. Consider the class of databases containing a monadic predicate P; further restrict the class so that every database has at least one definite tuple for P. Now the query $\exists x P(x)$ has the definite answer property for any database in this class. However, the query $P(x)$ does not have the representative model property for this class. Consider the database $D = \{P(c), P(o)\}$ where $Dom(o) = \{a, b\}$. Neither of its models serves as a representative model. This is because each of the two models entails one of $P(a)$ or $P(b)$ but neither of these atoms is entailed by D. However, this class of databases is very unnatural, and one may ask if the representative model property is entailed by the definite answer property for the particular classes of OR-databases that we are interested in, namely the classes of databases defined by some OR-database schema. The answer does not seem to be straightforward and we leave it as an open question. However, we note that this question is not germane to our current pursuit.

4 Polynomial Time Queries on Type I Databases

In this section we identify the class of queries which have polynomial time data complexity with respect to type I databases. Specifically, we define a notion of *strong cyclicity* and show that every query that is *not* strongly cyclic has polynomial time data complexity. We first show that queries which are not strongly cyclic have the representative model property. The polynomial time data complexity result follows from this.

Definition 4.1 (Strongly Cyclic)
A query Φ is strongly cyclic if it contains distinct literals l_P and l_Q which strongly mark each other.

> **Example 4.1 (Strongly cyclic queries)**
> The query $\exists xy[P^{A,O}(x,y) \wedge Q^{O,A}(x,y)]$ is strongly cyclic. The literal l_P marks the literal l_Q via the path $\langle y \rangle$. Similarly, l_Q marks l_P via $\langle x \rangle$.
>
> On the other hand, the query $\Phi \equiv \exists xyz[P^{A,O,A}(x,y,z) \wedge Q^{A,A}(y,z) \wedge R^{O,A}(x,z)]$ is not strongly cyclic. While l_R strongly marks l_P, the converse is not true: all paths from l_P to a variable of l_R pass through the variable z, which occurs in l_P.

Note that in the above example, the query Φ which is not strongly cyclic contains a literal l_R which is not strongly marked by any other literal. The following lemma generalizes this observation.

Lemma 4.1 *Any query which is not strongly cyclic contains a literal which is not strongly marked.*

Proof: Suppose that Φ is not strongly cyclic, but that every literal of Φ is strongly marked. We show that a contradiction follows. Since every literal of Φ is strongly marked, it follows that there exists a sequence of literals $l_0, l_1, \ldots, l_n$ such that for $i = 0 \ldots n-1$ the literal l_i strongly marks the literal l_{i+1}, and l_n strongly marks l_0. Take the shortest such sequence.

If $n = 1$ then we have two mutually strongly marking literals, contradicting the assumption that the query is not cyclic. If $n > 1$ then applying Lemma 3.2 to the literals l_0, l_1 and l_2, we obtain that either l_1 strongly marks l_0, contradicting the assumption that the query is not strongly cyclic, or that l_0 strongly marks l_2, yielding a shorter cycle of strong markings, and contradicting the minimality assumption. ∎

We now establish that queries which are not strongly cyclic have the representative model property for type I databases. The proof leads immediately to a polynomial time algorithm for answering this class of queries. We first state some conventions that will be crucial to the correct interpretation of the proof and algorithm to follow.

> **Conventions**
> We wish to speak of "the OR-objects of an atom". This will be done according to the convention that if τ is an atom containing no OR-objects, then the empty vector Λ is the vector of OR-objects of τ.

In the following, $\vec{o}$ denotes a vector of OR-objects and $Dom(\vec{o})$ denotes the cartesian product of their respective domains. We use the convention that $Dom(\Lambda) = \{\Lambda\}$.

If τ is an atom containing the OR-objects $\vec{o}$, then $\tau[\vec{d}/\vec{o}]$ denotes the result of substituting values $\vec{d}$ for the OR-objects $\vec{o}$. Thus, if $\tau = P(c, o_1, o_2)$, then $\tau[\langle a, b\rangle / \langle o_1, o_2\rangle] = P(c, a, b)$. Note that a and b are members of $Dom(o_1)$ and $Dom(o_2)$, respectively. If τ contains no OR-objects then $\tau[\Lambda/\Lambda] = \tau$.

Lemma 4.2 (Representative model property in type I databases)
Let $\Phi(\vec{x})$ be such that the free variables $\vec{x}$ are not strongly marked in Φ and Φ is not strongly cyclic. Then $\Phi(\vec{x})$ has the representative model property for type I databases, and for each type I database D it is possible to compute in polynomial time a substitution θ such that $D\theta$ is a representative model of D with respect to $\Phi(\vec{x})$.

Proof: We need to show that for each type I database D there exists a valuation θ_R such that $m = D\theta_R$ is a representative model, i.e.,

$$\{ \vec{a} \mid m \models \Phi(\vec{a})\} \;=\; \{ \vec{a} \mid D \models \Phi(\vec{a})\}.$$

In the case that Φ is the null query containing no literals, the result is trivial. Assume, for a proof by induction, that the result holds for all queries shorter than Φ. Since $\Phi(\vec{x})$ is not strongly cyclic it follows from Lemma 4.1 that some literal l_P of Φ is not strongly marked. Therefore we may write

$$\Phi(\vec{x}) = \exists \vec{y}\ [l_P(\vec{x}, \vec{y}) \wedge \Phi_1(\vec{x}, \vec{y})].$$

Since l_P is not strongly marked, none of the variables $\vec{x}$ or $\vec{y}$ is strongly marked in the query Φ_1. Further, since Φ_1 is a subquery of Φ it is not strongly cyclic. Let $D = D_P \cup D_1$ be the decomposition of D into the relation P and the remaining relations. It follows from the induction hypothesis that there exists a valuation θ_1 for the OR-objects of D_1 such that

$$\{ \langle \vec{a}, \vec{b}\rangle \mid D_1\theta_1 \models \Phi_1(\vec{a}, \vec{b})\} \;=\; \{ \langle \vec{a}, \vec{b}\rangle \mid D_1 \models \Phi_1(\vec{a}, \vec{b})\}.$$

We extend θ_1 to the required valuation θ_R so that $m = D\theta_R$ is a representative model of D for $\Phi(\vec{x})$. Let $\tau \in D_P$ be an atom of P with OR-objects $\vec{o}$. Suppose the following condition *fails*:

$$\begin{array}{l}\text{For every } \vec{d} \in Dom(\vec{o}) \text{ there exist some } \vec{b} \text{ such that}\\ \text{(i) } \tau[\vec{d}/\vec{o}] = l_P(\vec{a}, \vec{b}) \text{ and (ii) } D_1 \models \Phi_1(\vec{a}, \vec{b}).\end{array} \qquad (1)$$

Then τ does not contribute $\vec{a}$ to the output of $\Phi(\vec{x})$ on D, i.e., $(\{\tau\} \cup D_1) \not\models \Phi(\vec{a})$. Hence, in our representative model, $\tau\theta_R$ must also not contribute $\vec{a}$ to the output of $\Phi(\vec{x})$ on D, i.e., $(\{\tau\} \cup D_1)\theta_R \not\models \Phi(\vec{a})$. This is achieved by putting $\theta_R(\vec{o}) = \vec{d}$ for some $\vec{d}$ that witnesses the failure of (1).

On the other hand, if (1) holds, then τ necessarily contributes $\vec{a}$ to the output of $\Phi(\vec{x})$ on D, i.e., $(\{\tau\} \cup D_1) \models \Phi(\vec{a})$. Hence, in our representative model, $\tau\theta_R$ will contribute $\vec{a}$ to the output of $\Phi(\vec{x})$, i.e., $(\{\tau\} \cup D_1)\theta_R \models \Phi(\vec{a})$, for any choice of $\vec{d}$. Therefore, in this case we choose $\vec{d}$ in $Dom(\vec{o})$ arbitrarily.

¿From the above construction it follows that if $(\{\tau\} \cup D_1)\theta_R \models \Phi(\vec{a})$ then $(\{\tau\} \cup D_1) \models \Phi(\vec{a})$. Also, note that the construction crucially depends on the fact that D is a type I database. Otherwise, different atoms containing the same OR-object o may require different values for $\theta_R(o)$.

We now claim that $m = D\theta_R$ is such that $\{\vec{a} \mid D \models \Phi(\vec{a})\} = \{\vec{a} \mid m \models \Phi(\vec{a})\}$. Since m is a model of D, obviously $\{\vec{a} \mid D \models \Phi(\vec{a})\} \subseteq \{\vec{a} \mid m \models \Phi(\vec{a})\}$. For the converse, suppose that $m \models \Phi(\vec{a})$. Then for some $\tau \in D_P$, $(\{\tau\} \cup D_1)\theta_R \models \Phi(\vec{a})$. By the construction this implies that $(\{\tau\} \cup D_1) \models \Phi(\vec{a})$. Hence, $D \models \Phi(\vec{a})$. ∎

We note that according to the conventions, condition (1) simplifies to

There exist some $\vec{b}$ such that (i) $\tau = l_P(\vec{a}, \vec{b})$ and (ii) $D_1 \models \Phi_1(\vec{a}, \vec{b})$

if τ contains no OR-objects. The following example illustrates the construction used in the above proof.

Example 4.2 (Computing a representative model)
Consider query $\Psi(x) = \exists y[P^{A,A,O}(x,x,y) \wedge Q^{A,A}(x,y)]$. Note that the free variable of the query is *not* strongly marked (but it is marked). Also note that the literal $l_P(x,y)$ is not strongly marked in $\Psi(x)$. Let D be an OR-database as shown in Figure 4.2. We construct the valuation θ_R such that $D\theta_R$ is the desired representative model.

Following the construction of the above proof, we write $\Psi(x) = \exists y[l_P(x,y) \wedge \Psi_1(x,y)]$ where $\Psi_1(x,y) = Q(x,y)$, and the database D as $D = D_P \cup D_1$ where D_P is the set of atoms of P and D_1 is the set of atoms of Q.

The Database D

$P(x,x,y)$	$Q(x,y)$
$P(a,a,o_1)$	$Q(a,1)$
$P(b,c,o_2)$	$Q(a,2)$
$P(d,d,o_3)$	$Q(d,3)$

Domains

$Dom(o_1) = \{1,2\}$
$Dom(o_2) = \{1,2\}$
$Dom(o_3) = \{3,4\}$

A Representative Model $\mathbf{m_R}$

$P(x,x,y)$	$Q(x,y)$
$P(a,a,1)$	$Q(a,1)$
$P(b,c,1)$	$Q(a,2)$
$P(d,d,4)$	Q(d,3)

Figure 4.2: A database, query and a representative model

There are three kinds of atoms of D_P: (i) those which in conjunction with D_1 necessarily contribute to the output of $\Psi(x)$ (ii) those that never contribute to the output of $\Psi(x)$ and (iii) those which in conjunction with D_1 may or may not contribute to the output of $\Psi(x)$.

Consider the atom $\tau_1 = P(a, a, o_1)$. This atom necessarily contributes "a" to the output of $\Psi(x)$ on D. This is because, for any $d \in Dom(o_1)$, the result $P(a, a, d)$ of substituting d for o_1 in the atom $P(a, a, o_1)$ has a matching atom $Q(a, d)$, thus making $\Psi(a)$ true. Hence "a" is in the output of $\Psi(x)$ on D. In other words, the condition (1) holds for the atom τ_1. Hence, the representative model can choose any value for o_1.

Consider the second atom $\tau_2 = P(b, c, o_2)$. This atom never contributes to the output of $\Psi(x)$ on D. This is because, $\tau_2[1/o_2] = P(b, c, 1)$ and $\tau_2[2/o_2] = P(b, c, 2)$, and neither of these unifies with $l_P(x, y)$ because their first and second arguments are not the same. So, we can choose any value from $Dom(o_2)$ in the representative model. Here, the condition (1) fails because (i) fails for all values of d.

Finally, the atom $\tau_3 = P(d, d, o_3)$ may or may not contribute to the output of $\Psi(x)$. For, if we assign 3 to o_3, then there is a matching atom $Q(d, 3)$ whereas if we assign 4 to o_3 there is no matching atom $Q(d, 4)$. Following the proof, we choose the value 4 for o_3 to construct θ_R. The desired representative model is shown in Figure 4.2.

The classification of closed queries with polynomial time evaluation with respect to type I databases is a corollary of Lemma 4.2. Note that the proof actually shows that $D \models \Phi(\vec{a})$ if and only if condition (1) holds for some $\tau \in D_P$. This yields the recursive algorithm shown in Figure 2. To apply this algorithm to a closed query we take $\vec{x}$ to be a null vector of variables. Recall from Section 3 that the output of a closed query is either the empty set (indicating the answer "no") or the set $\{\Lambda\}$ containing the null vector Λ (indicating the answer "yes").

It is easy to verify that the complexity of the algorithm is in PTIME. Hence we have the following theorem.

Theorem 4.3 (Type I databases and PTIME evaluation)
If Φ is a closed query which is not strongly cyclic, then Φ has polynomial time data complexity with respect to type I databases.

5 Polynomial Time Queries on Type II Databases

In this section we turn our attention to the data complexity of queries on type II databases, providing a class of queries with polynomial time data complexity. In the case of type II and type III databases it is convenient to establish this classification first for restricted classes of queries, and then extend it to the full class of queries.

Procedure Entails-I($\Phi(\vec{x}), D$) :

% D : a type I database.
% $\Phi(\vec{x}) = \exists\vec{y}[l_P(\vec{x},\vec{y}) \wedge \Phi_1(\vec{x},\vec{y})]$ a non-strongly cyclic query
% with variables $\vec{x}, \vec{y}$ not strongly marked in Φ_1.
% Returns the set of tuples of constants $\{\vec{a} | D \models \Phi(\vec{a})\}$

Begin

$A \leftarrow$ Entails-I($\Phi_1(\vec{x},\vec{y}), D$);
Return($\{\vec{a}$ | there exists $\tau \in D_P$ with OR-objects $\vec{o}$ such that for every $\vec{d} \in Dom(\vec{o})$ there exist some $\vec{b}$ satisfying (i) $\tau[\vec{d}/\vec{o}] = l_P(\vec{a},\vec{b})$ and (ii) $\langle\vec{a},\vec{b}\rangle \in A\}$)

End

Figure 2: PTIME algorithm for strongly acyclic queries on type I databases

5.1 Normal Queries

We first introduce normal queries and then we show how a classification of these queries can be "lifted" to the full set of proper queries. Next we show how to identify polynomial time normal queries.

Definition 5.1 *A proper query is normal if each literal has at most one OR-argument.*

We first show how to obtain a normal query that corresponds to any given query Φ.

Let $l = P^{T_1,\ldots,T_n}(x_1,\ldots,x_n)$ be a literal, where the variables x_i are not necessarily distinct and each ornation T_i is either A or O. We define the *split* of the literal to be the query

$$split(l) = \exists u[P_1^{T_1,A}(x_1,u) \wedge \ldots \wedge P_n^{T_n,A}(x_n,u)]$$

with free variables $x_1 \ldots x_n$. If Φ is the query $\exists\vec{x}[l_1 \wedge \ldots \wedge l_m]$ then we define the split of Φ by

$$split(\Phi) = \exists\vec{x}[split(l_1) \wedge \ldots \wedge split(l_m)].$$

Observation 5.1 *Given any query Φ, the query $split(\Phi)$ is normal.*

Now we show how a classification of the complexity of the normal query $split(\Phi)$ extends to the original query Φ. For this we need to define a notion of "reducibility". This notion is a variation of the *reducibility* used in classical complexity theory [3].

Definition 5.2 (Reducibility)
A query Φ is reducible to another query Φ' over a class $\mathcal{C}$ of databases, denoted by $\Phi \leq_{\mathcal{C}} \Phi'$, iff there exists a polynomial time function $f : \mathcal{C} \longrightarrow \mathcal{C}$ such that for any database $D \in \mathcal{C}$, $D \models \Phi$ iff $f(D) \models \Phi'$. Furthermore, if $\Phi \leq_{\mathcal{C}} \Phi'$ and $\Phi' \leq_{\mathcal{C}} \Phi$, then we write $\Phi \equiv_{\mathcal{C}} \Phi'$.

We write $\Phi \leq_I \Phi'$, $\Phi \leq_{II} \Phi'$ and $\Phi \leq_{III} \Phi'$ for reduction over type I, type II and type III databases respectively.

We first show how $Split(\Phi)$ and Φ are related to each other through reducibility.

Lemma 5.2 *For every query Φ, $\Phi \equiv_{II} split(\Phi)$ and $\Phi \equiv_{III} split(\Phi)$.*

Proof: To see that $\Phi \leq_{II} split(\Phi)$ and $\Phi \leq_{III} split(\Phi)$, define the mapping f from databases D in the predicates of Φ to databases in the predicates of $split(\Phi)$ as follows. For each atom $P(u_1, \ldots, u_n) \in D$, where the u_i may be either constants or OR-objects, choose a *new* constant c, and add to the database $f(D)$ the atoms $P_1(u_1, c)$, $\ldots$,$P_n(u_n, c)$. It is readily verified that $f(D)$ may be computed in time polynomial in the size of D, that $f(D)$ is a type II database if D is of type II, and that $D \models \Phi$ if and only if $f(D) \models split(\Phi)$.

To see that $split(\Phi)$ reduces to Φ for both type II and type III databases, let $f(D)$ be the database which contains the atom $P(u_1, \ldots, u_n)$ whenever there exists a constant c such that the database D contains all the atoms $P(u_1, c)$, $\ldots$, $P(u_n, c)$. Again $f(D)$ is computable in polynomial time, is a type II database if D is of type II, and $D \models \Phi$ if and only if $f(D) \models split(\Phi)$. ∎

Observation 5.3 (Splitting)
From the inequality $\Phi \leq_{\mathcal{C}} split(\Phi)$ it follows that if $split(\Phi)$ has PTIME data complexity for databases in the class $\mathcal{C}$ then so does Φ. On the other hand, it follows from $split(\Phi) \leq_{\mathcal{C}} \Phi$ that if $split(\Phi)$ has co-NP hard data complexity for databases in $\mathcal{C}$, then so does Φ.

Thus, by Lemma 5.2, a characterization of the complexity of normal (proper) queries yields a characterization of all (proper) queries, for type II and type III databases. Thus it suffices to consider normal queries for type II and type III databases.

We comment that Lemma 5.2 does not hold for type I databases. Consider $\Phi = \exists x[P^{O,O}(x, x)]$. This query is not strongly cyclic and hence has PTIME data complexity (from Theorem 4.3). However, $split(\Phi) = \exists xy[P_1^{A,O}(y, x) \wedge P_2^{A,O}(y, x)]$ has co-NP complete data complexity (shown in Section 9). Thus splitting does not preserve complexity for type I databases. That is why, for type I databases, we developed a characterization that applies directly to *all* proper queries.

5.2 Polynomial Time Normal Queries

In this section we obtain a syntactic characterization of PTIME normal queries for type II databases. The notion of representative model is again crucial to this characterization. We define a class of queries called "acyclic queries" and show (in Lemma 5.5) that acyclic queries have the representative model property for type II databases and that the representative model can be computed in time that is a polynomial in the size of the input database. It follows from the proof of this that the data complexity of these queries is in PTIME for type II databases.

In Section 4 we showed that if the query $\Phi(\vec{x})$ is not strongly cyclic and the free variables $\vec{x}$ are not strongly marked, then it has the definite answer property in type II databases. We have already seen in Example 3.2 that this result does not extend to type II databases. Thus, the class of queries with the representative model property in type II databases is a strict subset of the corresponding class for type I databases, the non-strongly cyclic queries. The following definition helps to describe this smaller set.

Definition 5.3 (Acyclic Query)
The normal query Φ is *cyclic* if there exists distinct literals l_P and l_Q in Φ which mark each other. Otherwise it is acyclic.

The query

$$\exists xyz\ [P^{A,O,A}(x,y,z) \wedge Q^{A,A}(y,z) \wedge R^{O,A}(x,z)]$$

is cyclic. The literal l_R marks l_P through the path $\langle x \rangle$ and l_P marks l_R through the path $\langle y, z \rangle$. Note that we showed in Example 4.1 that this query is not strongly cyclic. Thus the strongly cyclic queries form a proper subset of the cyclic queries.

For type I databases we know from Lemma 4.2 that if $\Phi(\vec{x})$ is not strongly cyclic and the free variables $\vec{x}$ are not strongly marked then $\Phi(\vec{x})$ has the representative model property. Similarly, for type II databases *acyclic* queries will be shown to have the representative model property with respect to their *unmarked* variables. There are some further differences, however. Recall that in the type I case we showed that Φ can be written as $\Phi(\vec{x}) = \exists \vec{y}[l_P(\vec{x},\vec{y}) \wedge \Phi_1(\vec{x},\vec{y})]$ where l_P is not strongly marked. Next, we showed that $\Phi(\vec{x})$ has the representative model property for type I databases by showing how to construct a representative model for any type I database by considering one atom at a time of the relation P. (See Example 4.2 for an illustration.) However, in general this approach does not work for type II databases, because OR-objects may be shared by two or more atoms.

Example 5.1 (Shared OR-objects)
Consider the query

$$\Phi(x) = \exists yz[P^{A,O}(y,z) \wedge Q^{A,A}(y,z) \wedge R^{A}(x)]$$

which is strongly acyclic. The variable x is unmarked in this query, which may be shown to have the representative model property. In the type I case, to show that $D \models \Phi(c)$ it suffices to find a single atom $P(a,o) \in D$ such that $D \models Q(a,d) \wedge R(c)$ for all $d \in Dom(o)$. This is no longer true if D is a type II database. For example, let D be the database

$$\{P(a,o), P(b,o), Q(a,1), Q(b,0), R(c)\}$$

where $Dom(o) = \{0,1\}$. Here we have neither $Q(a,d) \in D$ for all $d \in Dom(o)$ nor $Q(b,d) \in D$ for all $d \in Dom(o)$, but still $D \models \Phi(c)$. This is because, if o is assigned 0, the atoms $P(b,0)$, $Q(b,0)$ and $R(c)$ provide the desired witness, while if o is assigned 1, the atoms $P(a,1)$, $Q(a,1)$ and $R(c)$ provide the desired witness.

On the other hand, there are other queries for which it does turn out to be possible to consider tuples individually with respect to the contribution they make to the answer in a representative model.

Example 5.2 (Tuple at a Time in Type II)
Consider the strongly acyclic query

$$\Psi(x) = \exists yz[P^{A,A,O}(x,y,z) \wedge Q^{A,A}(x,y) \wedge R^{A}(z)]$$

in which x is unmarked. Let D be the database

$$\{P(c_1,a,o), P(c_2,b,o), P(c_3,a,o), Q(c_1,a), Q(c_2,b), R(0), R(1)\}.$$

In this case, we have $D \models \Psi(c_1)$ and $D \models \Psi(c_2)$ but not $D \models \Psi(c_3)$. For this query, it does turn out to be possible to consider the tuples of P one at a time. Intuitively, this is because the variables x and y are disconnected from z in the remainder $Q(x,y) \wedge R(z)$ of the the query. Note that some atoms $P(e,f,u)$ can never contribute to the answer, because there is no tuple $Q(e,f)$. The tuple $P(c_3,a,o)$ is of this kind. For the remaining atoms $P(e,f,u)$, we have an atom $Q(e,f)$ in D. Hence, depending on whether the value d chosen for u corresponds to a tuple $R(d)$, these atoms all contribute to the answer, or none do. For the database D, the tuples $P(c_1,a,o)$ and $P(c_2,b,o)$ are of this sort.

These two queries, it turns out, provide a canonical set of examples. The following result, analogous to Lemma 4.1, shows that acyclic queries with unmarked free variables are of one of two forms. The proof is deferred to Appendix A.

Lemma 5.4 *Let $\Phi(\vec{x})$ be a normal acyclic query with unmarked free variables $\vec{x}$. If not all columns of Φ are ATOMIC then Φ contains an unmarked literal l_P which has an OR-argument, and may be written in one of the two forms*

$$\exists \vec{y} z[l_P(\vec{x_1}, \vec{y}, z) \wedge \Psi_1(\vec{x_1}, \vec{x_2}, \vec{y}) \wedge \Psi_2(z)]$$

or

$$\Psi_1(\vec{x}) \wedge \exists \vec{y} z[l_P(\vec{y}, z) \wedge \Psi_2(\vec{y}, z)]$$

where $\vec{x} = \vec{x_1}\vec{x_2}$ *and* z *is the variable in the OR-argument of* l_P.

Using this decomposition of queries, it is possible to obtain a result corresponding to Lemma 4.2. The proof is very similar, and appears in Appendix A.

Lemma 5.5 (Representative model property in type II databases)
Let $\Phi(\vec{x})$ *be a normal query with free variables* $\vec{x}$. *Suppose that* Φ *is not cyclic and that the variables* $\vec{x}$ *are not marked in* Φ. *Then* $\Phi(\vec{x})$ *has the representative model property for type II databases, and for each type II database* D *it is possible to compute in polynomial time a substitution* θ *such that* $D\theta$ *is a representative model of* D *with respect to* $\Phi(\vec{x})$.

The class of queries in polynomial time with respect to type II databases is obtained from Lemma 5.5, by taking $\vec{x}$ to be an empty set of variables. As with Lemma 4.2 the proof of Lemma 5.5 yields a polynomial time algorithm, shown in Figure 3. As before, we use the convention that constants c occurring in OR-arguments are treated as trivial OR-objects.

Theorem 5.6 *If* Φ *is a closed query which is normal and acyclic then* Φ *has polynomial time data complexity with respect to type II databases.*

6 Polynomial Time Queries on Type III Databases

In this section we obtain a classification of queries that have PTIME data complexity for type III databases. We first obtain a classification for *connected* normal queries and then show how to extend this to all normal queries. Then from Lemma 5.2 and Observation 5.1 it follows that we have a classification for all proper queries.

Definition 6.1 (Connected Queries)
A query Φ *is connected if for every pair* x, y *of variables of* Φ *there exists a sequence* $\langle x_1, \ldots, x_n \rangle$ *of variables such that* $x_1 = x$, $x_n = y$ *and for each* $i = 1 \ldots n-1$ *some literal of* Φ *contains both variables* x_i *and* x_{i+1}. *A connected component of a query* Φ *is a maximal connected subset of the set of literals of* Φ.

Every query can be written as the conjunction of its connected components. For example, the query

$$\exists xyzuv[P(x, y) \wedge Q(x, z) \wedge R(u, v) \wedge S(u)]$$

Procedure Entails-II$(\Phi(\vec{x}), D)$:

%

% D a type II database.

% $\Phi(\vec{x})$ a normal acyclic query with $\vec{x}$ unmarked

% Returns the set of tuples of constants $\{ \vec{a} \mid D \models \Phi(\vec{a})\}$

%

If $\Phi(\vec{x})$ contains no OR-arguments then use a standard query evaluation method

Else if $\Phi(\vec{x_1}, \vec{x_2}) \equiv \exists \vec{y} z [l_P(\vec{x_1}, \vec{y}, z) \wedge \Psi_1(\vec{x_1}, \vec{x_2}, \vec{y}) \wedge \Psi_2(z)]$

where $\vec{x} = \vec{x_1}\vec{x_2}$, the literal l_P is unmarked with z the variable in the OR-argument then

Begin

$A_1 \leftarrow$ Entails-II$(\Psi_1(\vec{x_1}, \vec{x_2}, \vec{y}), D)$;

$A_2 \leftarrow$ Entails-II$(\Psi_2(z), D)$;

Return($\{\langle \vec{a_1}, \vec{a_2}\rangle \mid$ there exists an atom $\tau = l_P(\vec{a_1}, \vec{b}, o) \in D_P$ containing OR-object o

such that $\langle \vec{a_1}, \vec{a_2}, \vec{b}\rangle \in A_1$

and for all $d \in Dom(o)$, $d \in A_2\}$)

End

Else if $\Phi(\vec{x}) \equiv \Psi_1(\vec{x}) \wedge \exists \vec{y} z [l_P(\vec{y}, z) \wedge \Psi_2(\vec{y}, z)]$

with literal l_P unmarked and z the variable in the OR-argument then

Begin

$A_1 \leftarrow$ Entails-II$(\Psi_1(\vec{x}), D)$;

$A_2 \leftarrow$ Entails-II$(\Psi_2(\vec{y}, z), D)$;

If there exists an OR-object o such that

for all $d \in Dom(o)$ there exists an atom $\tau \in D_P$

containing o and there exist constants $\vec{b}$

with $\tau[d/o] = l_P(\vec{b}, d)$ and $\langle \vec{b}, d\rangle \in A_2$

then Return(A_1)

else Return($\emptyset$)

End

Figure 3: PTIME Algorithm for Normal Acyclic Queries on type II databases

may be written as

$$\exists xyz[P(x,y) \wedge Q(x,z)] \wedge \exists uv[R(u,v) \wedge S(u)].$$

Since we are dealing with proper queries, this allows us to reduce the problem of classifying the complexity of a query to the problem of classifying the complexity of its connected components. Thus, if all the connected components of a query Φ have PTIME data complexity, then so does Φ. Conversely, if some component of Φ has co-NP hard data complexity, then Φ also has co-NP hard data complexity. Hence, without loss of generality we consider only connected queries. Furthermore, from Lemma 5.2 it follows that we need to consider only normal queries (i.e. queries with at most one OR-argument per literal). Thus, we restrict our attention to connected normal queries.

The following notion of redundant arguments helps us in obtaining the desired classification.

Definition 6.2 (Redundant Argument and Redundant Column)
An argument labeled by x is redundant in a query Φ if x does not occur anywhere else in Φ. If D is a database and Φ a query, then the columns of D corresponds to the redundant arguments of Φ are referred to as redundant columns.

For example the query $\exists xyz[P(x,y) \wedge Q(x,z)]$ has the arguments labeled by y and z redundant, but the arguments labeled by x are not redundant. The significance of the notion of redundancy is that redundant OR-arguments do not contribute to the data complexity of queries. This is already apparent from our classifications for type I and type II databases: a literal containing only redundant OR-arguments cannot mark another literal.

The representative model property again plays an important role in proving polynomial time complexity. However, unlike the type I and type II proofs, the PTIME algorithm does not follow immediately from this property. The following result provides a class of queries having the representative model property for type III databases.

Lemma 6.1 *The query $\Phi(\vec{x})$ with unmarked free variables $\vec{x}$ has the representative model property for type III databases if it has at most one irredundant OR-argument.*

Proof: Let Φ' denote the query obtained by deleting all the variables labeling redundant OR-arguments of Φ. Since an OR-argument is always marked (by definition), it follows that none of the arguments labeled by $\vec{x}$ are deleted from Φ in forming Φ'. Given a database D, construct the database D' by deleting the redundant OR-columns from D. It is readily verified that $D \models \Phi(\vec{a})$ if and only if $D' \models \Phi'(\vec{a})$. Since D' is trivially type II (D' has at most one OR-column), Lemma 5.5 ensures that D' has a representative model $m_{D'}$. Thus,

$$\{\ \vec{a} \mid D \models \Phi(\vec{x})\} \ \equiv \ \{\ \vec{a} \mid D' \models \Phi'(\vec{x})\} \ \equiv \ \{\ \vec{a} \mid m_{D'} \models \Phi'(\vec{x})\}$$

Domains

$Dom(u_1) = \{2, 3\}$
$Dom(u_2) = \{2, 3\}$

The Database D

$P_1^{A,O}(x,y)$		$Q_1^A(y)$	$P_2^{A,O}(x,z)$		$Q_2^A(z)$
k_1	2	2	k_1	u_1	3
k_2	u_1		k_2	u_2	
k_3	u_2		k_3	3	

Figure 4: A type III database D and the domains of the OR-objects

Clearly any extension of $m_{D'}$ to a model of D (by assigning to each OR-object an arbitrary member of its domain) will suffice as a representative model. ∎

Note that queries with at most one irredundant OR-argument are necessarily acyclic. The following example shows that Lemma 6.1 cannot be generalized to acyclic connected queries with more than one irredundant OR-argument.

Example 6.1 (Irredundancy and the Representative Model Property)
Consider the query

$$\Phi(x) = \exists yz[P_1^{A,O}(x,y) \wedge Q_1^A(y) \wedge P_2^{A,O}(x,z) \wedge Q_2^A(z)].$$

This is an acyclic query in which the variable x is unmarked. It has two irredundant OR-arguments. By Lemma 5.5, this query has the representative model property with respect to type II databases. We now demonstrate that this property no longer holds when we move to type III databases. The database D given in Figure 4 illustrates this. It is easy to verify that

$$(i)\ D \models \bigvee_{i=1}^{3} \Psi(k_i)\ but\ (ii)\ for\ \ 1 \leq i \leq 3,\ D \not\models \Psi(k_i)$$

Suppose m is a representative model of D. From (i) it follows that $m \models \Psi(k_i)$ for some i. Since m is a representative model, $D \models \Psi(k_i)$, contradicting (ii). Therefore D does not have a representative model.

For the proof of the following theorem we again use the convention that constants in OR-arguments are treated as trivial OR-objects.

Theorem 6.2 *A normal query has PTIME data complexity on type III databases if it is acyclic and no connected component has more than two irredundant OR-arguments.*

Proof: By the observation above it suffices to prove that if Φ is a connected normal acyclic query with at most two irredundant OR-arguments then Φ has PTIME data complexity. If Φ has only redundant OR-arguments then the result is immediate from Lemma 6.1. Otherwise, by Lemma 5.4, we may write

$$\Phi = \exists \vec{x}[l_P(\vec{x}) \wedge \Phi_1(\vec{x})]$$

where l_P has a irredundant OR-argument and the variables $\vec{x}$ are unmarked in Φ_1. As usual we write D_P for the atoms of D with predicate P.

The intuition for the algorithm described below is as follows. Given a type III database D, we attempt to construct a counter-model to Φ. Suppose that D_P contains an atom τ with OR-object o and let $d \in Dom(o)$. Let $\vec{a}$ be constants satisfying (i) $l_P(\vec{a}) = \tau[d/o]$ and (ii) $D \models \Phi_1(\vec{a})$. Then clearly any valuation assigning the value d to o will not produce a countermodel. Thus we may eliminate such d as candidate values for o by removing them from $Dom(o)$. Because D_P shares OR-objects with other relations, this may result in a database for which $\Phi_1(\vec{x})$ has a larger set of definite answers. This may enable additional values to be removed from the domains of OR-objects in D_P using the consideration above. We repeat this process until no more deletions are possible. We show that if no object has empty domain at the end of this process then the countermodel exists. Figure 5 formally describes the algorithm. To establish correctness of the algorithm and also prove that it runs in time polynomial in the size of the database D we note the following.

First, for any i, if $D^i \models \Phi$ then $D^{i+1} \models \Phi$. This is because the set of models of D^{i+1} is a subset of the set of models of D^i. The same consideration shows that $L^i \subset L^{i+1}$ for all i, so the algorithm cannot run for more than a polynomial number of iterations. The query Φ_1 has at most one irredundant OR-argument, hence by Lemma 6.1 the set L^i can be computed in polynomial time. It follows that the algorithm runs in polynomial time.

Next, if $D^i \not\models \Phi$ then $D^{i+1} \not\models \Phi$. For, suppose that $D^i \not\models \Phi$. Then there exists a valuation η of D^i such that $D^i\eta \not\models \Phi$. We show that for every $o \in V_P$, $\eta(o) \in Dom^{i+1}(o)$ and hence η is also a valuation of D^{i+1}. This implies that $D^{i+1} \not\models \Phi$. Let o be an arbitrary member of V_P and suppose $\eta(o)$ is one of the values removed from Dom^i to obtain Dom^{i+1}. Then there exists an atom $\tau \in D_P$ containing o and there exist constants $\vec{a}$ with $\tau\eta = l_P(\vec{a})$ and $D^i \models \Phi_1(\vec{a})$. But this implies that $D^i\eta \models l_P(\vec{a}) \wedge \Phi_1(\vec{a})$, contradicting $D^i\eta \not\models \Phi$. Hence $\eta(o)$ is not removed from $Dom^i(o)$ to obtain $Dom^{i+1}(o)$. We have now shown that for every i, $D^i \models Q$ iff $D^{i+1} \models Q$. Since $D^0 = D$ we have that $D \models \Phi$ if and only if $D^i \models \Phi$. It follows from this that when the algorithm returns "yes", it is indeed the case that $D \models Q$.

Finally we show that if $L^i = L^{i+1}$ then $D \not\models Q$. This implies that when the algorithm returns "no", we in fact have $D \not\models Q$. Suppose $L^i = L^{i+1}$. We construct a valuation θ for D^{i+1} such that $D^{i+1}\theta \not\models \Phi$. Define the set

$$W = \{ \ \langle \vec{a} \rangle \ \mid \ \tau[d/o] = l_P(\vec{a}) \text{ for some } \tau \in D_P \text{ with OR-object } o \text{ and for some } d \in Dom^{i+1}(o)\}$$

Procedure Entails-III(Φ, D)

% D a type III database, $\Phi = \exists\vec{x}[l_P(\vec{x}) \wedge \Phi_1(\vec{x})]$ a normal connected query
% with no redundant OR-arguments and at most two irredundant
% OR-arguments, one of them in l_P
% Returns "yes" iff $D \models \Phi$.

Initializations:

$V_P = \{\ o \mid o$ is an OR-object which occurs in $D_P\}$
$D^0 = D$
$L^0 =$ Entails-II($\Phi_1(\vec{x}, D)$ % this equals $\{\langle\vec{a}\rangle \mid D \models \Phi_1(\vec{a})\}$
$i := 0$

Repeat the following steps:

(Step 1). **if** there exists $o \in V_P$ such that for every $d \in Dom^i(o)$
there exists an atom $\tau \in D_P$ with OR-object o
and there exist constants $\vec{a}$ such that
$\tau[d/o] = l_P(\vec{a})$ and $\langle\vec{a}\rangle \in L^i$
then return("yes").

(Step 2). For every $o \in V_P$, modify the domains as follows:
$Dom^{i+1}(o) = Dom^i_1(o) - \{\ d \mid$ there exists an atom $\tau \in D_P$
and constants $\vec{a}$ such that
$l_P(\vec{a}) = \tau[d/o]$ and $\langle\vec{a}\rangle \in L^i\ \}$

(Step 3) Let D^{i+1} be the database obtained from D
by replacing the domain function Dom by Dom^{i+1}

(Step 4) $L^{i+1} =$ Entails-II($\Phi_1(\vec{x}), D^{i+1}$) % this equals $\{\langle\vec{a}\rangle \mid D^{i+1} \models \Phi_1(\vec{a})\ \}$

(Step 5) **if** $L^{i+1} = L^i$ **then return**("no")
else put $i := i + 1$ and **goto** step 1.

Figure 5: PTIME Algorithm for type III databases

We claim that

$$D^{i+1} \not\models \Phi_1(\vec{a}) \text{ for all } \langle\vec{a}\rangle \in W. \quad (1)$$

For, suppose that $\langle\vec{a}\rangle \in W$. Then there exists an atom τ containing OR-object o, and there exists $d \in Dom^{i+1}(o)$ such that $\tau[d/o] = l_P(\vec{a})$. This means that $\langle\vec{a}\rangle \notin L^i$, else d would have been deleted from $Dom^i(o)$. Since $L^{i+1} = L^i$, we have that $\langle\vec{a}\rangle \notin L^{i+1}$, in other words, $D^{i+1} \not\models \Phi_1(\vec{a})$, proving (1). Since Φ_1 has the representative model property, we may choose a valuation θ of D^{i+1} such that $D^{i+1}\theta \models \Phi_1(\vec{a})$ implies $D^{i+1} \models \Phi_1(\vec{a})$. We claim that $D^{i+1}\theta \not\models \Phi$, from which $D \not\models \Phi$ follows. For this, suppose that to the contrary, $D^{i+1}\theta \models \Phi$. Then for some $\langle\vec{a}\rangle \in W$ we have $D^{i+1}\theta \models \Phi_1(\vec{a})$, so $D^{i+1} \models \Phi_1(\vec{a})$. But this contradicts (1). ∎

Note that in the above proof it is crucial that Φ_1 has the representative model property. We showed that if Φ_1 has two or more connected irredundant OR-arguments, it fails to have this property and hence we can not generalize the above proof for arbitrary Φ_1. In fact, it turns out that if Φ_1 has two connected irredundant OR-arguments, i.e., if Φ has three connected and irredundant OR-arguments, the query is co-NP hard.

7 Tools for Hardness Proofs

In this section we introduce the techniques which will be used in our proofs that various classes of queries have co-NP complete data complexity. Our approach to the hardness results will be as follows. We first identify a small set of "basic" queries, which are shown to be co-NP complete. Next, we use certain transformations which simplify the structure of queries to show that for each query Φ in the class under consideration there exists a basic query Φ_B with a polynomial time reduction (see Definition 5.2) $\Phi_B \leq_{\mathcal{C}} \Phi$ over the class $\mathcal{C}$ of databases. Since Φ_B is co-NP hard for the class of databases $\mathcal{C}$, it then follows that Φ also is co-NP hard for the class $\mathcal{C}$.

The same transformations are used in the context of type I, type II and type III databases. What will vary is the way in which they are applied. Each of the transformations matches the structure of the query in certain ways, then modifies the query according to this match. If there is no way to perform the pattern match the transformation is inapplicable. In the following definitions of the transformations the notation $T : L \rightarrow R$ signifies that the transformation T applied to a query matching the pattern L is transformed into the query R.

$$T_1 : \exists x\vec{z}\,[P^{d,\vec{d}}(x,\vec{z}) \wedge \Psi] \longrightarrow \exists\vec{z}\,[P^{\vec{d}}(\vec{z}) \wedge \Psi] \qquad \text{where } x \text{ has a single occurrence in the input.}$$

$$T_2 : \exists xy\vec{w}\,[P^{A,A,\vec{d}}(x,y,\vec{w}) \wedge \Psi] \longrightarrow \exists x\vec{w}\,[\{P^{A,\vec{d}}(x,\vec{w}) \wedge \Psi\}[x/y]\,]$$

$$T_3 : \exists x[P^A(x) \wedge \Psi] \longrightarrow \exists x[\Psi]$$

$$T_4 : \exists x[P^{O,A}(x,x) \wedge \Psi] \longrightarrow \exists x[P^O(x) \wedge \Psi].$$

Here the notation $\Phi[x/y]$ denotes the query obtained by substituting x for each occurrence of y in a literal of Φ. Note that the applicability of these transformations depends on the typing of arguments in l_P, except in the case of T_1 which applies regardless of the type of the argument labeled by x. The notation $\vec{d}$ indicates that the typing of those arguments are of no concern ("don't care").

Each of these transformations "deletes" one of the arguments of the query. The remaining arguments preserve their ornations. The order of the arguments in the literal P and the order of literals in the query are immaterial to the applicability of a transformation. The following demonstrates a sequence of applications of the transformations:

$$\exists xyut\ [P^{O,A,A}(x,y,t) \wedge Q^{A,A}(x,y) \wedge R^{A,O}(u,t)]$$

$$\longrightarrow^{T_2}\ \exists xut\ [P^{O,A,A}(x,x,t) \wedge Q^{A}(x) \wedge R^{A,O}(u,t)]$$

$$\longrightarrow^{T_1}\ \exists xt\ [P^{O,A,A}(x,x,t) \wedge Q^{A}(x) \wedge R^{O}(t)]$$

$$\longrightarrow^{T_3}\ \exists xt\ [P^{O,A,A}(x,x,t) \wedge R^{O}(t)]$$

$$\longrightarrow^{T_2}\ \exists x\ [P^{O,A}(x,x) \wedge R^{O}(x)]$$

$$\longrightarrow^{T_4}\ \exists x\ [P^{O}(x) \wedge R^{O}(x)]$$

We now establish some properties of these transformations. First, applying any of these transformations to a query does not increase its data complexity. Second, the transformations preserve marking relationships.

Lemma 7.1 *For every query Φ, if T is one of the transformations T_1, T_2 , T_3 or T_4 then $T(\Phi) \leq_{II} \Phi$ and $T(\Phi) \leq_{III} \Phi$. If T is one of the transformations T_1, T_2 or T_3 then $T(\Phi) \leq_{I} \Phi$.*

Proof: For each transformation T, we show how to construct from a database D another database $f(D)$ such that $D \models T(\Phi)$ iff $f(D) \models \Phi$. In each case $f(D)$ will differ from D only in the relation P. Verification that $f(D)$ indeed has the required property will be straightforward, and is left to the reader.

For T_1, let $f(D)$ contain an atom $P(a,\vec{b})$ for every atom $P(\vec{b})$ of D, where a is a fixed constant. For T_2, let $P(a,a,\vec{b})$ be an atom of $f(D)$ whenever $P(a,\vec{b})$ is an atom of D. For T_3, we let $f(D)$ contain an atom $P(a)$ for every constant a appearing in D. For T_4, for each atom $P(o)$ in D, where o is an OR-object with domain $Dom(o) = \{a_1,\ldots,a_k\}$ we let $f(D)$ contain atoms $P(o,a_1),\ldots,P(o,a_k)$. In each case the database $f(D)$ can be constructed in polynomial time and is of type II if D is of type II. If T is one of the transformations T_1, T_2 or T_3 then we also have that $f(D)$ is a type I database if D is a type I database. ∎

We note that the construction of this proof does not work for type I databases in the case of the transformation T_4, because here the database $f(D)$ may contain multiple occurrences of OR-objects, even if D does not.

Lemma 7.2 *Let T be one of the transformations $T_1 - T_4$. Suppose Q and R are predicates of Φ not deleted by the transformation T and let l_Q, l_R and l'_Q, l'_R be the literals with predicates Q, R in Φ and $T(\Phi)$ respectively. If l_Q marks l_R in Φ then l'_Q marks l'_R in $T(\Phi)$.*

Proof: Let $\sigma = \langle x_0, \ldots, x_n \rangle$ be a marking path from l_R to l_Q of minimal length. Then x_n occurs in l_R and we have the following properties:

1. the variable x_0 occurs in an OR-argument of l_Q, and
2. if $n > 0$ then Φ contains a literal (possibly equal to l_Q) which contains both x_1 and an occurrence of the variable x_0 distinct from the occurrence of x_0 used to satisfy condition 1, and
3. $x_i \neq x_0$ for $i = 1 \ldots n$, and
4. for all $i = 1 \ldots n-1$ there exists a literal in Φ containing both variables x_i and x_{i+1}.

The minimality assumption implies that every x_k has at least two occurrences. For x_0 this is immediate from condition 2. If x_n has only one occurrence then this must be in l_R, hence x_{n-1} is in l_R also, contradicting the minimality assumption. Suppose x_k occurs in a single literal for some k with $0 < k < n$. Consider the segment $\langle \ldots, x_{k-1}, x_k, x_{k+1}, \ldots \rangle$ of σ. By conditions 2 and 4 of the definition of marking above, there is a literal, l_{k-1}, in which x_{k-1} and x_k appear and similarly, there is a literal l_k in which x_k and x_{k+1} appear. Since x_k appears in only one literal, $l_{k-1} = l_k$, and this literal contains both x_{k-1} and x_{k+1}. Thus, we obtain a shorter marking path $\langle x_0 \ldots, x_{k-1}, x_{k+1}, \ldots x_n \rangle$, violating the minimality assumption. Hence, we can assume without loss of generality that each variable in σ appears in Φ at least twice.

Call the literals used to satisfy conditions 2 and 4 of the definition of marking *path literals*. A similar argument may be used to show that each path literal contains at least two distinct variables. We now show how each of the transformations "preserve" the marking path.

Consider $T_1 : \exists x \vec{z}\, [P^{d,\vec{d}}(x, \vec{z}) \wedge \Psi] \longrightarrow \exists \vec{z}\, [P^{\vec{d}}(\vec{z}) \wedge \Psi]$, where x has a single occurrence in the input. Clearly, the variable x cannot be any of the variables in σ because each of these has at least two occurrences. Hence σ is a marking path from l'_Q to l'_R in $T_1(\Phi)$.

Consider $T_3 : \exists x\ [P^A(x) \wedge \Psi] \longrightarrow \exists x\ [\Psi]$. Here, the literal l_P cannot be any of the path literals, because these contain at least two distinct variables. (Note also that l_P cannot be either l_Q or l_R because we assumed the predicates Q and R are not deleted by the transformation.) Hence σ remains a marking path after an application of T_3.

Consider $T_4 : \exists x\ [P^{O,A}(x, x) \wedge \Psi] \longrightarrow \exists x\ [P^O(x) \wedge \Psi]$. Here the literal l_P cannot be a path literal because these have at least two distinct variables. The literal l_P may be l_Q or l_R. In the former case, σ remains a marking path in $T(\Phi)$ because the literal used to satisfy condition 2 cannot be l_P. In the latter case σ remains a a marking path because the variable $x_n = x$ still appears in l'_R.

Finally, consider $T_2 : \exists xy\vec{w}\ [P^{A,A,\vec{d}}(x, y, \vec{w}) \wedge \Psi] \longrightarrow \exists x\vec{w}\ [\{P^{A,\vec{d}}(x, \vec{w}) \wedge \Psi\}[x/y]]$. If y does not appear in σ, it is clear that σ remains a marking path. Suppose that y appears in σ: say, $\sigma = \langle x_0, \ldots, x_{m-1}, y, x_{m+1}, \ldots, x_n\rangle$. Let x_{m-1} and y appear in l_{m-1}; similarly, y and x_{m+1} appear in l_m. After applying T_2, x_{m-1} and x appear in l_{m-1} while x and x_{m+1} appear in l_m. We now have two cases. If $x \neq x_0$ then $\langle x_0, \ldots, x_{m-1}, x, x_{m+1}, \ldots, x_n\rangle$ is a marking path from l'_Q to l'_R. If $x = x_0$ then $\langle x_0, x_{m+1}, \ldots, x_n\rangle$ is a marking path from l'_Q to l'_R. A similar argument applies if $y = x_0$ or $y = x_n$ ■

Note that none of the transformations deletes an OR-argument that marks any other literal. Thus we have the following immediate consequence of Lemma 7.2.

Corollary 7.3 *If Φ is cyclic then $T(\Phi)$ is cyclic, where T is one of the transformations $T_1 - T_4$.*

We now exhibit the basic co-NP Complete cyclic queries which are used to establish co-NP hardness in the type I and type II case. The following problem is useful to establish these queries.

Definition 7.1 (Monotone-3SAT)
Let A be a set of propositional variables. A literal *is either a propositional variable $a \in A$, in which case it is said to be* positive, *or an expression $\bar{a}$ for some $a \in A$, in which case it is said to be* negative. *A* clause *is a set of literals. A set $\mathcal{C}$ of clauses is said to be* satisfiable *if there exists a function $V : A \rightarrow \{true, false\}$, such that for each clause $c \in \mathcal{C}$, either there exists $a \in c$ with $V(a) = true$ or there exists $\bar{a} \in c$ with $V(a) = false$. A clause is said to be 3-monotone if it has exactly three literals, all of which are positive or all of which are negative. The problem MONOTONE-3SAT (M-3SAT) is the collection of satisfiable sets of 3-monotone clauses.*

It is well known [3] that M-3SAT is NP complete. We will write sets of 3-monotone clauses as $\mathcal{C} = \mathcal{P} \cup \mathcal{N}$ where $\mathcal{P}$ is the set of positive clauses of $\mathcal{C}$ and $\mathcal{N}$ the set of negative clauses. The following notion will be useful in establishing the next lemma.

Definition 7.2 (Clause-Mapping)
A clause-mapping *of a set $\mathcal{C}$ of monotone clauses is a mapping $\alpha : \mathcal{C} \rightarrow A$ where A is the set of*

propositional variables appearing in $\mathcal{C}$, such that for all $c \in \mathcal{C}$, $\alpha(c) \in c$ if c is a positive clause, and $\alpha(c) = a$ for some $\bar{a} \in c$ if c is a negative clause. A clause-mapping is inconsistent *if there exists a positive clause $c \in \mathcal{P}$ and a negative clause $d \in \mathcal{N}$ such that $\alpha(c) = \alpha(d)$.*

It is not difficult to see that a set $\mathcal{C}$ of monotone clauses is not in M-3SAT if and only if every clause-mapping of $\mathcal{C}$ is inconsistent.

Lemma 7.4 *The following queries have co-NP hard data complexity with respect to type I databases.*

$$\Phi_1 = \exists x[P^O(x) \wedge Q^O(x)]$$

$$\Phi_2 = \exists xy[P^{O,A}(x,y) \wedge Q^{O,A}(y,x)]$$

Proof: We show that there exists a reduction from M-3SAT to these queries. That is, we show that there exists a polynomial time function f from sets $\mathcal{C} = \mathcal{P} \cup \mathcal{N}$ of monotone clauses to type I databases, such that $\mathcal{C} \notin$ M-3SAT if and only if $f(\mathcal{C}) \models \Phi$.

For Φ_1, the function f acts as follows. We write D for $f(\mathcal{C})$. For each positive clause $c = \{a_1, a_2, a_3\}$ we introduce a *new* OR-object o_c with $Dom(o_c) = \{a_1, a_2, a_3\}$ and add the atom $P(o_c)$ to D. Similarly, for each negative clause $c = \{\bar{a}_1, \bar{a}_2, \bar{a}_3\}$, we introduce a new OR-object o_c with $Dom(o_c) = \{a_1, a_2, a_3\}$ and add the atom $Q(o_c)$ to D. Clearly, f is a polynomial time function. Notice that the valuations of D correspond exactly to the clause mappings of $\mathcal{C}$, via the equation $\alpha(c) = \theta(o_c)$. Now $\mathcal{C} \notin$ M-3SAT exactly when all clause mappings of $\mathcal{C}$ are inconsistent. By the correspondence, this holds exactly when for all valuations θ of D we have $D\theta \models \Phi_1$, that is, if and only if $D \models \Phi_1$.

For Φ_2, the database $D = f(\mathcal{C})$ is constructed as follows. For each positive clause $c = \{a_1, a_2, a_3\}$ we introduce a *new* OR-object o_c with $Dom(o_c) = \{a_1, a_2, a_3\}$ and add the atom $P(o_c, true)$ to D. Similarly, for each negative clause $c = \{\bar{a}_1, \bar{a}_2, \bar{a}_3\}$, we introduce a new OR-object o_c with $Dom(o_c) = \{a_1, a_2, a_3\}$ and add the atom $P(o_c, false)$ to D. For each propositional variable a appearing in $\mathcal{C}$, we introduce a new OR-object u_a with $Dom(u_a) = \{true, false\}$ and add the atom $Q(u_a, a)$ to D. It is clear that f is a polynomial time function.

Suppose that $\mathcal{C} \notin$M-3SAT. Then every clause-mapping of $\mathcal{C}$ is inconsistent, so in every model of D there exists a propositional variable a such that $P(a, true)$ and $P(a, false)$ both belong to D. Since every model of D also contains either $Q(true, a)$ or $Q(false, a)$, it follows that every model of D satisfies Φ_2. This shows that $\mathcal{C} \notin$ M-3SAT implies $D \models \Phi_2$. Conversely, suppose that $\mathcal{C} \in$ M-3SAT. Then there exists a consistent clause mapping $\alpha : \mathcal{C} \rightarrow A$. Let θ be the valuation defined by $\theta(o_c) = \alpha(c)$ and $\theta(u_a) = false$ if there exists a positive clause $c \in \mathcal{P}$ with $\alpha(c) = a$ and $\theta(u_a) = true$ otherwise. We claim that $D\theta \not\models \Phi_2$, which implies that $D \not\models \Phi_2$. For, suppose that $D\theta \models \Phi_2$. We show there are two cases, both contradictory. First, we may have that there exists atoms $P(a, true), Q(true, a) \in D\theta$. But $P(a, true) \in D\theta$ implies that there exists a positive clause

c with $\alpha(c) = a$. It follows from this that $\theta(u_a) = false$, so that $Q(true, a) \notin D$, a contradiction. Alternatively, there exist atoms $P(a, false), Q(false, a) \in D\theta$. The first of these implies that there exists a negative clause $c \in \mathcal{N}$ with $\alpha(c) = a$. The second implies that there exists a positive clause $d \in \mathcal{P}$ with $\alpha(d) = a$. But this contradicts the consistency of α. ■

8 Co-NP Hard Queries on Type II Databases

We saw in Section 5 that every acyclic query has polynomial time data complexity on type II databases. In this section we complete the classification for type II databases by showing that every cyclic query has co-NP complete data complexity on type II databases.

Theorem 8.1 *Every cyclic query has co-NP hard data complexity on type II databases.*

Proof: Let Φ be a cyclic query. We show that $\Phi_B \leq_{II} \Phi$ for one of the basic cyclic queries Φ_B of Lemma 7.4. Since Φ is cyclic, there exist two literals l_P and l_Q such that l_P marks l_Q and l_Q marks l_P. Let Φ_0 be the query obtained by setting all arguments other than the two that witness these markings between l_P and l_Q to be ATOMIC. Clearly $\Phi_0 \leq_{II} \Phi$. Now apply T_1 through T_4 to Φ_0 repeatedly, until a query Φ' is produced which can not be further reduced. This process stops because each subsequent query produced has one fewer argument. By Lemma 7.1, we have that $\Phi' \leq_{II} \Phi$. Now we show that for some basic query Φ_B, it is the case that $\Phi_B \leq \Phi'$, establishing the co-NP hardness of Φ' and hence Φ.

The transformation T_2 ensures that Φ' has at most one ATOMIC argument in any literal. The transformation T_3 ensures that no literal in Φ' has just one ATOMIC argument. Hence every literal in Φ' has exactly one ATOMIC argument and one OR-argument, or just one OR-argument. Thus the query Φ' has exactly two literals, because Φ_0 has two OR-arguments only. Because of T_1 no variable occurs just once in the query. By Corollary 7.3 we must have that Φ' is cyclic. It follows that Φ' must have one of the following forms:

Case (i): The two OR-arguments are labeled by the same variable and both literals have one ATOMIC argument:

$$\Phi' = \exists xu[P_1^{O,A}(x, u) \wedge P_2^{O,A}(x, u)].$$

In this case it is easy to prove that $\Phi_B \leq_{II} \Phi'$ where

$$\Phi_B = \exists x[P_1^O(x) \wedge P_2^O(x)].$$

Case (ii): The two OR-arguments are labeled by the same variable, and neither literal has an ATOMIC argument:

$$\Phi' = \exists x[P_1^O(x) \wedge P_2^O(x)].$$

which itself is a basic query.

Case (iii): The two OR-arguments are labeled by different variables:

$$\Phi_B = \exists xy[P_1^{O,A}(x,y) \wedge P_2^{O,A}(y,x)].$$

which again is a basic query itself. Thus Φ has co-NP hard data complexity. ∎

9 Co-NP Hard queries on Type I Databases

In Section 4 we showed that all strongly acyclic queries have polynomial time complexity on type I databases. We now show that the rest of the queries have co-NP hard complexity.

The proof of hardness for type II databases permitted an arbitrary sequence of applications of the transformations of Section 7. In the context of type I databases we need to be more careful about how we apply the transformations, since it may happen that a query with co-NP hard complexity is transformed in to one with PTIME complexity. For example

$$\exists xyt\ [P^{O,A,A}(x,y,t) \wedge Q^{A,A,A}(x,y,z) \wedge R^{O,A}(t,z)]$$

is strongly cyclic and has co-NP hard complexity, whereas applying transformation T_2 to the literal l_Q produces the query

$$\exists xtz\ [P^{O,A,A}(x,x,t) \wedge Q^{A,A}(x,z) \wedge R^{O,A}(t,z)]$$

which is strongly acyclic, hence has PTIME complexity.

Theorem 9.1 *The data complexity of any strongly cyclic query on type I databases is co-NP hard.*

Proof: If Φ is strongly cyclic, there exist two distinct literals l_P and l_Q such that l_P strongly marks l_Q and l_Q strongly marks l_P. Let $\sigma_1 = \langle u_0, u_1, \ldots, u_k \rangle$ be the shortest strong marking path from l_P to l_Q and let $\sigma_2 = \langle v_0, v_1, \ldots, v_m \rangle$ be the shortest strong marking path from l_Q to l_P. Let Φ_0 be the query obtained from Φ by setting all the OR-arguments except those labeled by u_0 in l_P and by v_0 in l_Q to be ATOMIC. Clearly, $\Phi_0 \leq_I \Phi$. We show that Φ_0 is co-NP hard and hence the result follows. There are two cases: either σ_1 and σ_2 have a variable in common, or they do not. We show that in either case, Φ_0 is co-NP hard.

Case I: Suppose that σ_1 and σ_2 share a variable, say $u_i = v_j$. First note that by the definition of strong marking, the variables $u_1, \ldots, u_k$ do not occur in l_P and the variables $v_1, \ldots, v_m$ do not occur in l_Q. Also, the variables u_0 through u_{k-1} do not occur in l_Q since otherwise a marking path shorter than σ_1 can be chosen. Similarly, v_0 through v_{m-1} do not occur in l_P. Write the path $\langle u_0, u_1, ..., u_i, v_{j-1}, ..., v_1, v_0 \rangle$ as $\langle w_0, w_1, \ldots, w_n \rangle$. Then we have the following:

(i) For all $i = 1 \dots n-1$, the variable w_i appears in neither l_P nor l_Q.

(ii) For all $i = 0 \dots n-1$ there exists a literal l_i containing both variables w_i and w_{i+1} in ATOMIC arguments.

Now apply the transformation T_2 repeatedly to the literals l_0 through l_{n-1}. When applying the transformation to l_i, we will have that l_i contains the variables u_0 and w_{i+1}, so we delete the argument labeled by w_{i+1} and substitute u_0 for each occurrence of w_{i+1} elsewhere in the query. Since none of the variables $w_1, \dots w_n$ appear in l_P or l_Q, and the variables $u_0 = w_0$ and $v_0 = w_{n+1}$ have only single occurrences in the OR-arguments of l_P and l_Q respectively, the result of this sequence of transformations will be a query of the form

$$\Phi_1 = \exists u_0 \vec{z}\vec{t}\, [l_P(u_0, \vec{z}) \wedge l_Q(u_0, \vec{t}) \wedge \Psi(u_0, \vec{z}, \vec{t})]$$

where u_0 appears in each of l_P and l_Q only in the OR-argument. It is straightforward to show that the basic co-NP hard query $Q_B = \exists x[P^O(x) \wedge Q^O(x)]$ is such that $Q_B \leq_I \Phi_1$. It follows by Lemma 7.1 that $\Phi_B \leq \Phi$ and hence Φ is also co-NP hard.

Case (ii): Suppose the marking paths are disjoint. In this case we apply the transformation T_2 repeatedly along the path $\langle u_0, u_1, ..., u_k\rangle$ and then repeatedly along the path $\langle v_0, v_1, ..., v_m\rangle$ to obtain a query of the form

$$\Phi_1 = \exists u_0 v_0 \vec{z}\vec{t}\, [l_P(u_0, v_0, \vec{z}) \wedge l_Q(v_0, u_0, \vec{t}) \wedge \Psi(u_0, v_0, \vec{z}, \vec{t})]$$

where u_0 occurs in l_P only in the single OR-argument and v_0 appears in l_Q only in its single OR-argument. It is now easy to show that $\Phi_B \leq_I \Phi_1$ where Φ_B is the basic co-NP hard query $\exists xy\ [P^{O,A}(x, y) \wedge Q^{O,A}(y, x)]$. It follows that Φ is co-NP hard. ∎

10 Co-NP Hard Queries on Type III Databases

We now complete the classification of the complexity of queries on type III databases. In section 6 we showed that every connected acyclic query with two or fewer irredundant OR-arguments has PTIME data complexity. In this section we show that every connected query with three or more irredundant OR-arguments is co-NP complete. Our approach to proving the hardness result will be similar to that used for type I and type II databases above . We first identify a set of "basic" queries which may be shown to be hard on type III databases, and then prove that every connected query with three or more irredundant OR-arguments is co-NP hard by reduction from these basic queries.

In order to obtain the reduction of queries to one of the basic queries, we use the transformations of Section 7, with one minor modification. Define the transformation T_3' as follows:

$$T_3': \ \exists[P^A(x) \wedge \Psi] \longrightarrow \exists x[\Psi]$$

unless the variable x labels exactly one argument of Ψ, and this argument is an OR-argument.

That is, the transformation T_3' has exactly the same behavior as T_3, the only difference being that T_3' has a more stringent precondition. The effect of this precondition is to ensure that the transformation does not turn an irredundant OR-argument of Φ into a redundant OR-argument of $T_3'(\Phi)$. For example, the transformation T_3 applies to the query

$$\Phi = \exists x[P^O(x) \wedge Q^A(x)]$$

to produce the query $\exists x[P^O(x)]$ which has a redundant OR-argument. The transformation T_3' cannot be applied to Φ because its precondition is not met. It is straightforward to verify that each of the transformations T_1, T_2, and T_4 also has the property that any irredundant OR-argument of Φ remains an irredundant OR-argument of $T(\Phi)$.

Say that a query is *tri-connected* if it has a connected component with three or more irredundant OR-arguments. Then we have the following result.

Lemma 10.1 *Any tri-connected normal query is co-NP hard on type III databases.*

Proof: Without loss of generality we assume that Φ is connected. (We showed in section 5 that the data complexity of a query on type III databases is equal to the maximal data complexity of its connected components.) Let Φ_0 be the query produced from Φ by setting all but three irredundant OR-arguments to be ATOMIC. Clearly, $\Phi_0 \leq_{III} \Phi$. Apply the transformations T_1, T_2,T_3' and T_4 to Φ repeatedly until no further transformations are possible. Let Φ_B be the resulting query. If Φ_B is cyclic then it follows from Theorem 8.1 that Φ_B has co-NP hard data complexity on type III databases. Thus it suffices to consider the case of Φ_B acyclic.

It is readily verified that the transformations preserve connectedness, so since Φ is connected query, Φ_B is connected. As observed above, the transformations preserve irredundancy of OR-arguments, so Φ_B has exactly three OR-arguments, which are irredundant. Let P_1, P_2 and P_3 be the predicates containing OR-arguments. None of the P_i's have more than one ATOMIC argument because, then T_2 would have removed them. Hence each of the l_{P_i}'s can have at most one ATOMIC argument and exactly one OR-argument. The other literals, i.e., those that have no OR-arguments, referred to as Q_i's, can not have more than one ATOMIC argument due to T_2. Furthermore, the single argument in each Q_i should be labeled by a variable labeling one of the OR-arguments. Otherwise, such a Q_i will be removed by T_3'. Hence there can be at most three Q_i, each of which is monadic.

Finally, since Φ_B is acyclic, at least one l_{P_i} is not marked. Let it be l_{P_1}. Then the following cases are possible:

Case (i): l_{P_1} marks l_{P_2} and l_{P_2} marks l_{P_3}:

$$\Phi_B = \exists xyz[P_1^O(x) \wedge P_2^{A,O}(x,y) \wedge P_3^{A,O}(y,z) \wedge Q^A(z)].$$

Case (ii): l_{P_1} marks l_{P_2} and l_{P_3}:

$$\Phi_B = \exists xyz[P_1^O(x) \wedge P_2^{A,O}(x,y) \wedge Q_1^A(y) \wedge P_3^{A,O}(x,z) \wedge Q_2^A(z)].$$

Case (iii): l_{P_2} marks l_{P_3} only:

$$\Phi_B = \exists xyzt[P_1^{A,O}(x,y) \wedge Q_1^A(y) \wedge P_2^{A,O}(x,z) \wedge P_3^{A,O}(z,t) \wedge Q_2^A(t)].$$

Case (iv): No literal marks any other:

$$\Phi_B = \exists xyzt[P_1^{A,O}(x,y) \wedge Q_1^A(y) \wedge P_1^{A,O}(x,z) \wedge Q_2^A(z) \wedge P_2^{A,O}(x,t) \wedge Q_3^A(t)].$$

In every case, Φ_B is one of the queries which Lemma B.1 in Appendix B will show to be co-NP hard. It follows that Φ is co-NP hard also. ∎

This lemma immediately yields the following characterization of the queries with high data complexity on type III databases:

Theorem 10.2 *A normal query has co-NP hard data complexity on type III databases if it is either cyclic or tri-connected.*

11 Conclusions and Future Work

We have introduced in this paper the idea of complexity tailored database design, an approach to managing the intractability (co-NP hardness) of queries in databases containing incomplete information. Complexity tailored design involves setting schema parameters in such a way as to guarantee that the queries of interest to the user can be evaluated efficiently (with PTIME data complexity.) We have shown that for the class of proper queries in OR-databases, it is possible to characterize precisely the data complexity of each query with respect to a given database schema as being either PTIME or co-NP complete. This classification makes it possible (provided P $\neq$ Co-NP) to choose optimal values of the schema parameters so as to ensure that the query remains PTIME evaluable.

A number of extensions of our work are worth considering. First, our results need to be completed to give classification of the complexity of non-proper queries in OR-databases. We discuss in Appendix C the problems involved in such an extension, and give some partial results.

Secondly, we note that our present analysis is incomplete in that we have considered only closed queries. Our analysis is not immediately applicable to queries with free variables. Such an extension can be carried out in two different ways, since two semantics are plausible for queries containing free variables: one may return as answers to a query $\Phi(x)$ in a database D either

- all constants a such that $D \models \Phi(a)$, OR
- all (minimal) sets of constants $\{a_1, \ldots, a_n\}$ such that $D \models \Phi(a_1) \vee \ldots \vee \Phi(a_n)$.

Queries having the representative model property are interesting in this regard, since for such queries these two semantics coincide. Our classification does not apply to either extension. For example, consider the query $\Phi(x) \equiv P^O(x) \wedge Q^O(x)$, which may be shown to have the representative model property with respect to Type I databases. The results of this paper show that $\exists x \Phi(x)$ has co-NP complete data complexity for type I databases. However, the open query $\Phi(x)$ may be shown to have PTIME data complexity for type I databases. Thus, the classification developed in this paper does not apply to queries with free variables.

It may also be of interest to consider different schema parameters. For example, one could define a schema type which permitted coreference of OR-objects across certain pairs of columns, but not across others. Such extensions may permit a finer grained complexity tailored design, permitting maximal schemata which correspond to larger classes of databases than those of the present paper.

A final natural extension is to consider OR-objects whose domains are the entire set of constants in the database instance. These are the standard null values considered in [4]. Although our PTIME results hold for this extension, the hardness results do not hold. For example, $\exists x P^O(x) \wedge Q^O(x)$ can be shown to have PTIME data complexity for type I databases when the null values are allowed to take any constant from the database, instead of from a pre-specified set as in OR-objects. Recall that the same query has co-NP hard data complexity if the null values are OR-objects.

Appendices

A Representative Models of Type II Databases

In this appendix we give the detailed proof that acyclic queries have the representative model property for type II databases. (As a part of this we give a proof of Lemma 5.4 also.) This proof is very similar to Lemma 4.2, which dealt with non-strongly cyclic queries, but there are some differences. We remind the reader that we use the convention that constants c occurring in OR-arguments are treated as trivial OR-objects. That is, we replace each such occurrence, with a *new* OR-object u with $Dom(u) = \{c\}$.

Lemma 5.5: Let $\Phi(\vec{x})$ be a normal query with free variables $\vec{x}$. Suppose that Φ is not cyclic and that the variables $\vec{x}$ are not marked in Φ. Then $\Phi(\vec{x})$ has the representative model property for type II databases, and for each type II database D it is possible to compute in polynomial time a substitution θ such that $D\theta$ is a representative model of D with respect to $\Phi(\vec{x})$.

Proof: Assume, for a proof by induction, that the result holds for all queries shorter than Φ. Assume also, without loss of generality, that the query contains at least one OR-column. Since $\Phi(\vec{x})$ is not cyclic, by an argument much like that of Lemma 4.1 it follows that if Φ contains OR-arguments then we may write it in the form

$$\Phi(\vec{x_1}, \vec{x_2}) = \exists \vec{y} z [l_P(\vec{x_1}, \vec{y}, z) \wedge \Psi(\vec{x_1}, \vec{x_2}, \vec{y}, z)].$$

where $\vec{x} = \vec{x_1}\vec{x_2}$ and the variable z occurs in the OR-column of the unmarked literal l_P (there is exactly one OR-column because the query is normal). Notice that the variable z cannot be connected to any of the variables $\vec{x}$ in Ψ, since this would imply that some variable $\vec{x}$ is marked in $\Phi(\vec{x})$. We consider two cases:

CASE 1: Either the set of variables $\vec{x_1}$ is non-empty, or $\vec{x_1}$ is empty and some of the variables $\vec{x}$ are connected to some of the variables $\vec{y}$ in the query Ψ. Consider first the case that $\vec{x_1}$ is non-empty, i.e., some of the variables $\vec{x}$ occur in the atom l_P. In this case, the variable z cannot have a repeated occurrence in the atom l_P, since otherwise some of the variables $\vec{x}$ would be marked. Also, the variables $\vec{y}$ cannot be connected to z in the query Ψ, else the variables $\vec{x_1}$ would be marked in the query Φ. Hence, we may write

$$\Psi(\vec{x_1}, \vec{x_2}, \vec{y}, z) \equiv \Psi_1(\vec{x_1}, \vec{x_2}, \vec{y}) \wedge \Psi_2(z). \tag{1}$$

Next, consider the case that $\vec{x_1}$ is empty and some of the variables $\vec{x}$ are connected to some of the variables $\vec{y}$ in the query Ψ. Notice that this means that none of the variables $\vec{y}$ can be connected to z in Ψ, else once again some of the variables $\vec{x}$ are marked in Φ. Thus, in this case also we may decompose the query Ψ as in the equivalence (1). Note that we must also have once more that z does not occur twice in l_P, else some of the variables $\vec{x}$ are marked in Φ.

The variables $\vec{x}$ are not marked in Ψ_1 because they are not marked in Φ. The variables $\vec{y}$ are not marked in Ψ_1 because they all occur in the literal l_P, which is not marked in Φ. Similarly, the variable z is not marked in Ψ_2. Since Ψ_1 and Ψ_2 are subqueries of Φ they are not cyclic. Let $D = D_P \cup D_1 \cup D_2$ be the decomposition of D into the relation P, the relations in Ψ_1 and the relations in Ψ_2, respectively. It follows from the induction hypothesis that we may compute in polynomial time a valuation θ_1 of the OR-objects of D_1 such that $D_1\theta_1 \models \Psi_1(\vec{a_1}, \vec{a_2}, \vec{b})$ if and only if $D_1 \models \Psi_1(\vec{a_1}, \vec{a_2}, \vec{b})$. Similarly, we may compute in polynomial time a valuation θ_2 of the OR-objects in D_2 such that $D_2\theta_2 \models \Psi_2(d)$ if and only if $D_2 \models \Psi_2(d)$.

Let θ be the valuation defined on the OR-objects of D as follows. If the OR-object o occurs in D_1 then put $\theta(o) = \theta_1(o)$; if it occurs in D_2 then put $\theta(o) = \theta_2(o)$. If the OR-object o occurs in D_P and there exists a constant $d \in Dom(o)$ such that $D_2\theta_2 \not\models \Psi_2(d)$ then choose $\theta(o) = d$, else choose $\theta(o)$ to be an arbitrary element of $Dom(o)$. Note that this way of choosing $\theta(o)$ for o in D_P means that if $D_2\theta \models \Psi_2(\theta(o))$ then $D_2\theta_2 \models \Psi_2(d)$ for all $d \in Dom(o)$, hence $D_2 \models \Psi_2(d)$ for all $d \in Dom(o)$. The valuation θ can be constructed in time polynomial in the size of D.

We claim that the following are equivalent:

(i) $D \models \Phi(\vec{a_1}, \vec{a_2})$

(ii) $D\theta \models \Phi(\vec{a_1}, \vec{a_2})$

(iii) There exists an atom $\tau \in D_P$ containing OR-object o and there exist constants $\vec{b}$ such that $\tau = l_P(\vec{a_1}, \vec{b}, o)$ and $D_1 \models \Psi_1(\vec{a_1}, \vec{a_2}, \vec{b})$ and for all $d \in Dom(o)$, $D_2 \models \Psi_2(d)$.

Note that in order for this claim to be accurate it is important to treat tuples of D_P containing constants in the OR-argument according to the convention noted above, i.e., as containing trivial OR-objects. The implication from (i) to (ii) is immediate. To show (ii) implies (iii), suppose that $D\theta \models \Phi(\vec{a_1}, \vec{a_2})$. Then there exists a tuple $\tau \in D_P$ containing an OR-object o and there exist constants $\vec{b}$ such that $\tau\theta = l_P(\vec{a_1}, \vec{b}, \theta(o))$ and $D\theta \models \Psi_1(\vec{a_1}, \vec{a_2}, \vec{b}) \wedge \Psi_2(\theta(o))$. First, notice that because z occurs in l_P only in the OR-argument, it follows from $\tau\theta = l_P(\vec{a_1}, \vec{b}, \theta(o))$ that $\tau = l_P(\vec{a_1}, \vec{b}, o)$. Next, we have $D_1\theta_1 \models \Psi_1(\vec{a_1}, \vec{a_2}, \vec{b})$, so by construction of θ_1, we obtain $D_1 \models \Psi_1(\vec{a_1}, \vec{a_2}, \vec{b})$. Finally, $D_2\theta_2 \models \Psi_2(\theta(o))$, which implies that $D_2 \models \Psi_2(d)$ for all $d \in Dom(o)$, as noted above. This proves (iii). The implication from (iii) to (i) is straightforward.

CASE 2: The conditions of CASE 1 do not hold. That is, none of the variables $\vec{x}$ occur in the atom l_P and none of these variables are connected to any $\vec{y}$ in Ψ. Since the variables $\vec{x}$ cannot be connected to z, we may write

$$\begin{aligned} \Phi(\vec{x}) &\equiv \exists \vec{y} z [l_P(\vec{y}, z) \wedge \Psi_1(\vec{x}) \wedge \Psi_2(\vec{y}, z)] \\ &\equiv \Psi_1(\vec{x}) \wedge \exists \vec{y} z [l_P(\vec{y}, z) \wedge \Psi_2(\vec{y}, z)] \end{aligned}$$

In this case it is possible for the variable z to occur in l_P in an ATOMIC argument as well as in the OR-argument.

Let $D = D_1 \cup D_P \cup D_2$ be the decomposition of D into the relations in Ψ_1, the relation P and the relations in Ψ_2, respectively. Since Ψ_1, Ψ_2 are subqueries of Φ they are not cyclic. The variables $\vec{x}$ are not marked in Ψ_1 because they are not marked in Φ. It follows from the induction hypothesis that we may compute in polynomial time a valuation θ_1 of the OR-objects of D_1 such that $D_1\theta_1 \models \Psi_1(\vec{a})$ if and only if $D_1 \models \Psi_1(\vec{a})$. Similarly, the variables $\vec{y}$ and z are not marked in Ψ_2 because they all occur in the literal l_P, which is not marked in Φ. Thus it follows from the induction hypothesis that we may compute in polynomial time a valuation θ_2 of the OR-objects of D_2 such that $D_2\theta_2 \models \Psi_2(\vec{b}, d)$ if and only if $D_2 \models \Psi_2(\vec{b}, d)$.

Let θ be the valuation defined on the OR-objects of D as follows. If the OR-object o occurs in D_1 then put $\theta(o) = \theta_1(o)$; if it occurs in D_2 then put $\theta(o) = \theta_2(o)$. If the OR-object o occurs in D_P and there exists a constant $d \in Dom(o)$ such that

$$\begin{array}{l}\text{for all atoms } \tau \in D_P \text{ containing the OR-object } o \text{ and for all constants } \vec{b}, \\ \text{either } \tau[d/o] \neq l_P(\vec{b}, d) \text{ or } D_2\theta_2 \not\models \Psi_2(\vec{b}, d)\end{array} \tag{1}$$

then we put $\theta(o) = d$. Otherwise we choose for $\theta(o)$ an arbitrary element of $Dom(o)$. The valuation θ can be constructed in time polynomial in the size of D. We claim that the following are equivalent:

(i) $D \models \Phi(\vec{a})$

(ii) $D\theta \models \Phi(\vec{a})$

(iii) $D_1 \models \Psi_1(\vec{a})$ and there exists an OR-object o such that for all $d \in Dom(o)$ there exists an atom $\tau \in D_P$ containing o and there exist constants $\vec{b}$ such that $\tau[d/o] = l_P(\vec{b}, d)$ and $D_2 \models \Psi_2(\vec{b}, d)$.

Note that we once again rely on the convention that constants in OR-columns are treated as trivial OR-objects. The implication from (i) to (ii) is immediate. To show (ii) implies (iii), suppose that $D\theta \models \Phi(\vec{a})$. Then $D_1\theta_1 \models \Psi_1(\vec{a})$, so by construction of θ_1 we have $D_1 \models \Psi_1(\vec{a})$. Next, there exists a tuple $\tau \in D_P$ containing an OR-object o and there exist constants $\vec{b}$ such that $\tau\theta = l_P(\vec{b}, \theta(o))$ and $D_2\theta_2 \models \Psi_2(\vec{b}, \theta(o))$. Thus $d = \theta(o)$ does not satisfy the condition (1). By construction of θ, it follows that (1) holds for no constants $d \in Dom(o)$. That is, for all constants $d \in Dom(o)$ there exist a tuple $\tau \in D_P$ containing o and there exist constants $\vec{b}$ such that $\tau[d/o] = l_P(\vec{b}, d)$ and $D_2\theta_2 \models \Phi_2(\vec{b}, d)$. Since $D_2\theta_2 \models \Psi_2(\vec{b}, d)$ implies $D_2 \models \Psi_2(\vec{b}, d)$, this proves (iii). The implication from (iii) to (i) is straightforward. ∎

The correctness of the algorithm of Figure 3 follows from the equivalence of (i) and (iii) in both cases.

B co-NP hardness of the Basic Queries for Type III Databases

In this appendix we establish the hardness of the basic queries used in the proof of Lemma 10.1.

To understand the proof of the result to follow, let us briefly reconsider the proof of hardness for the query $\Gamma = \exists x[P^O(x) \wedge Q^O(x)]$, which worked by reduction from MONOTONE-3SAT (see Lemma 7.4).

> Each positive clause $c = \{a_1, a_2, a_3\}$ of a set of 3-clauses $\mathcal{C}$ is encoded as an atom $P(o_c)$ with $Dom(o_c) = \{a_1, a_2, a_3\}$ and similarly, each negative clause $c' = \{\overline{b_1}, \overline{b_2}, \overline{b_3}\}$ is encoded as an atom $Q(o_{c'})$ with $Dom(o_{c'}) = \{b_1, b_2, b_3\}$. The clause mappings of S correspond one-to-one to the models of the resulting database D. Next we note that a clause-mapping α is consistent if and only if the corresponding model M_α satisfies the query Γ.

Intuitively, we have used a fragment $D(c) = \{P(o_c)\}$ of the database D encoding the clause set to represent the positive clause c, and similarly for the negative clauses. More formally, an analysis of the proof shows that it relies on the following properties:
(i) For every valuation θ of $D(c)$, there exists an $a \in c$ such that $D(c)\theta \models P(a)$.
(ii) For each $a \in c$ there exists a valuation $\theta_{c,a}$ of $D(c)$ such that $D\theta_{c,a} \models P(x)$ if and only if $x = a$.
(iii) For no constant a and valuation θ is it the case that $D(c)\theta \models Q(a)$.
(iv) If for some valuation θ, $D\theta \models P(a)$ then there exists a clause c such that $D(c)\theta \models P(a)$.
The above conditions may be expressed by saying that the fragment $D(c)$ *represents the disjunction*

$$P(a_1) \bigvee P(a_2) \bigvee P(a_3)$$

Analogous properties hold for the negative clauses. The proof of the following result adapts these ideas to a more complex situation.

Lemma B.1 *The following acyclic queries have co-NP hard data complexity on type III databases:*

$$\Phi_1 = \exists xyzt \left[\begin{array}{l} P_1^{A,O}(x,y) \wedge Q_1^A(y) \wedge \\ P_2^{A,O}(x,z) \wedge P_3^{A,O}(z,t) \wedge Q_2^A(t) \end{array} \right]$$

$$\Phi_2 = \exists xyz \left[\begin{array}{l} P_1^O(x) \wedge \\ P_2^{A,O}(x,y) \wedge Q_1^A(y) \wedge \\ P_3^{A,O}(x,z) \wedge Q_2^A(z) \end{array} \right]$$

$$\Phi_3 = \exists xyz \left[\begin{array}{l} P_1^O(x) \wedge \\ P_2^{A,O}(x,y) \wedge P_3^{A,O}(y,z) \wedge Q^A(z) \end{array} \right]$$

$$\Phi_4 = \exists xyzt \left[\begin{array}{l} P_1^{A,O}(x,y) \wedge Q_1^A(y) \wedge \\ P_2^{A,O}(x,z) \wedge Q_2^A(z) \wedge \\ P_3^{A,O}(x,t) \wedge Q_3^A(t) \end{array} \right]$$

Proof: By reduction from MONOTONE-3SAT. Let $\mathcal{P}$ be a set of positive clauses and let $\mathcal{N}$ be a set of negative clauses over the set of atoms A. We will show how to construct for each query Φ a database D such that $D \models \Phi$ iff and only if the set of clauses $\mathcal{C} = \mathcal{P} \cup \mathcal{N}$ is unsatisfiable. In each case the proof of this fact will be based on a decomposition of the query Φ into two subqueries $\Psi_p(x), \Psi_n(x)$ such that Φ is equivalent to the query $\exists x(\Psi_p(x) \wedge \Psi_n(x))$. (This decomposition will not necessarily be based on a partition of the set of literals of the query: some literal of Φ may occur in both Ψ_p and Ψ_n.) We will write $\Phi(x)$ for $\Psi_p(x) \wedge \Psi_n(x)$. The positive clauses are represented using the query Ψ_p, and the negative clauses using Ψ_n.

In each case the database D will be the union

$$D = \bigcup_{c \in \mathcal{C}} D(c)$$

of a collection of databases $D(c)$ over the same schema. Intuitively, each database $D(c)$ represents the clause c. The domain over which $D(c)$ is based, i.e., the constants which appear in either the relations or the OR-objects of $D(c)$, will consist of the atoms appearing in c together with a *fresh* set of constants $Const(c)$. That is, if $c \neq d$ then $Const(c) \cap Const(d) = \emptyset$. Thus the intersection of the domains of $D(c)$ and $D(d)$ will be a subset of the set of atoms A. In addition, no OR-object o will occur in two distinct databases $D(c)$ and $D(d)$ where $c \neq d$. We write

$$Const = \bigcup_{c \in \mathcal{C}} Const(c).$$

It follows that that domain of D is $Const \cup A$.

Unfortunately, the properties (i)-(iii) discussed above cannot be obtained directly with these queries, so we need some slightly more complex conditions. In place of these properties the databases $D(c)$ will have the following *local* properties. For positive clauses $c \in \mathcal{P}$:

LP1: For all valuations θ, if $D(c)\theta \not\models \Phi$ then there exists $a \in c$ such that $D(c)\theta \models \Psi_p(a)$.

LP2: For each atom $a \in c$ there exists a valuation $\theta_{c,a}$ for the OR-objects of $D(c)$ such that (i) $D(c)\theta_{c,a} \not\models \Phi$ and (ii) if $x \in A$ is an atom, $D(c)\theta_{c,a} \models \Psi_p(x)$ if and only if $x = a$.

LP3: For no atom $a \in A$ and valuation θ is it the case that $D(c)\theta \models \Psi_n(a)$.

Notice that properties LP1, LP2, LP3 correspond to properties (i), (ii), (iii) respectively. These properties may be summarized by saying that instead of representing the disjunction $\Psi_p(a_1) \vee$

$\Psi_p(a_2) \vee \Psi_p(a_3)$, as in the simpler proof discussed above, the database $D(c)$ represents the disjunction $\Phi \vee \Psi_p(a_1) \vee \Psi_p(a_2) \vee \Psi_p(a_3)$. A similar set of properties will hold for the negative clauses: if $c \in \mathcal{N}$ then

LN1: For all valuations θ, if $D(c)\theta \not\models \Phi$ then there exists $\bar{a} \in c$ such that $D(c)\theta \models \Psi_n(a)$.

LN2: For each literal $\bar{a} \in c$ there exists a valuation $\theta_{c,a}$ for the OR-objects of $D(c)$ such that (i) $D(c)\theta_{c,a} \not\models \Phi$ and (ii) if $x \in A$ is an atom, $D(c)\theta_{c,a} \models \Psi_n(x)$ if and only if $x = a$.

LN3: For no atom $a \in A$ and valuation θ is it the case that $D(c)\theta \models \Psi_p(a)$.

The following Figures 6-9 show the queries Ψ_p, Ψ_n and how the databases $D(c)$ are constructed for each of the queries $\Phi_1 - \Phi_4$. In each case, we suppose that the clause c contains the atoms $\{a_1, a_2, a_3\}$. If $c \in \mathcal{P}$ then the database $D(c)$ is the union of a database $F(c)$ and a database $E_p(c)$. (In the case of Φ_2 and Φ_3 it suffices to take $D(c) = E_p(c)$ for positive clauses c, i.e., $F(c)$ is an empty set.) If $c \in \mathcal{N}$ then $D(c)$ is the union of $F(c)$ and $E_n(c)$. Note that for each clause c, we need to rename both the OR-objects and constants other than $\{a_1, a_2, a_3\}$ which appear in these tables, in order to obtain the required locality of constants and OR-objects.

Intuitively, the database $F(c)$ "creates a disjunction"

$$(D(c)\theta \models \Phi) \;\vee\; \theta(o_1) = 1 \;\vee\; \theta(o_2) = 1 \;\vee\; \theta(o_3) = 1 \tag{2}$$

which is then exploited in $E_p(c)$ to ensure $D(c) = F(c) \cup E_p(c) \models (\Phi \vee \Psi_p(a_1) \vee \Psi_p(a_2) \vee \Psi_p(a_1))$. Here the first disjunct of (2) accounts for the disjunct Φ, and if $\theta(o_i) = 1$ then $D\theta \models \Psi_p(a_i)$.The negative clauses are treated similarly. We now verify more formally that the databases $D(c)$ just described satisfy the properties required above.

LP1,LN1: Note that in each case, $F(c)$ is constructed in such a way that if $\theta(o_i) \neq 1$ for each $i = 1, 2, 3$, then $F(c)\theta \models \Phi$. (This is clear in the case of Φ_2 and Φ_3. For Φ_1 and Φ_4, see the first example of Section 6.) Now if $D(c)\theta \not\models \Phi$, then $F(c) \not\models \Phi$, so we must have $\theta(o_i) = 1$ for some i. It is then readily checked that $E_p(c)\theta \models \Psi_p(a_i)$ and $E_n(c)\theta \models \Psi_n(a_i)$.

LP2,LN2: In each case, $F(c)$ is constructed so that if θ_i is the valuation defined on o_1, o_2 and o_3 only, mapping exactly one of these to the value 1, then there exists an extension θ_{c,a_i} extending this valuation to the remaining OR-objects of $D(c)$ so that $F(c)\theta_{c,a_i} \not\models \Phi$. Furthermore, $D(c)\theta_{c,a_i} \models \Psi_p(a_j)$ if and only if $j = i$; an analogous result holds for Ψ_n.

LP3,LN3: These follow directly from the observation that the constants a_i occur in $E_p(c)$ and $E_n(c)$ only.

In representing a set of clauses as the union of databases $D(c)$, each of which represents a single clause c, we need to ensure that the effect of taking the union is additive. That is, we do not want interactions between the components to affect the way in which they represent clauses. In the simpler proof, the positive and negative clauses were represented by single atoms in separate relations, so there was no danger of interaction. This is formalized by the property (iv). In the present proof, however, the representation $D(c)$ of the clause c is spread out over a number of relations, and the relations used to represent positive clauses may overlap with those used to represent negative clauses. Furthermore, not all constants in the database correspond to atoms of the clause set. Hence it is slightly more complex to ensure that the effect of combining the databases $D(c)$ is additive. We check the following modifications of property (iv):

> **GP1:** If $a \in A$ is an atom and θ is a valuation for the OR-objects of D then $D\theta \models \Psi_p(a)$ if and only if there exists a clause $c \in \mathcal{P}$ such that $D(c)\theta \models \Psi_p(a)$.
>
> **GN1:** If $a \in A$ is an atom and θ is a valuation for the OR-objects of D then $D\theta \models \Psi_n(a)$ if and only if there exists a clause $c \in \mathcal{N}$ such that $D(c)\theta \models \Psi_n(a)$.
>
> **G2:** If $a \in Const$ and θ is a valuation for the OR-objects of D then $D\theta \models \Phi(a)$ if and only if there exists a clause $c \in \mathcal{C}$ such that $D(c)\theta \models \Phi(a)$.

These properties follow from the locality of constants to the $D(c)$. We show the argument for GP1 and Φ_1; the remaining cases are similar. Suppose that $D\theta \models \exists y[P_1(a_i, y) \wedge Q_1(y)]$. Then there exist tuples $P_1(a_i, o_i) \in D$ and $Q_1(k) \in D$ with $\theta(o_i) = k$. Let $c \in \mathcal{P}$ be the clause with $P_1(a_i, o_i) \in D(c)$. Then since $dom(o_i) = \{l_i, 1\}$ we have either $k = l_i$ or $k = 1$. But notice that $Q_1(l_i)$ is not a tuple of $D(c)$. Furthermore, by locality of constants, it is not a tuple of $D(d)$ for any clause $d \neq c$ either. It follows that $\theta(o_i) = 1$, from which $D(c) \models \Psi_p(a_i)$ is immediate.

We now show that it follows from the nine properties listed above that the set of clauses $\mathcal{C}$ is satisfiable if and only if $D \not\models \Phi$.

First, suppose that $\mathcal{C}$ is satisfiable, and let $\alpha : \mathcal{C} \rightarrow A$ be a clause-mapping witnessing this fact. We show how to construct a valuation θ from this clause-mapping such that $D\theta \not\models \Phi$. Since the OR-objects of D are local to the $D(c)$, we may define the valuation θ as a union of valuations on $D(c)$ by

$$\theta = \bigcup_{c \in \mathcal{C}} \theta_{c,\alpha(c)}.$$

where $\theta_{c,\alpha(c)}$ is a valuation of $D(c)$ satisfying LP2 and LN2. We claim that $D\theta \not\models \Phi$. For, suppose that $D\theta \models \Phi(x)$. We consider two cases: $x \in Const$ or $x \in A$. If $x \in Const$, then by G2 it follows that there exists a clause $c \in \mathcal{C}$ such that $D(c)\theta \models \Phi(x)$. By the definition of θ we have that $D(c)\theta_{c,\alpha(c)} \models \Phi(x)$. But this contradicts the fact that $\theta_{c,\alpha(c)}$ satisfies LP2(i) (or LN2(i), if c is a negative clause). If $x \in A$ then since $\Phi(x) \equiv \Psi_p(x) \wedge \Psi_n(x)$ we have $D\theta \models \Psi_p(x)$ and $D\theta \models \Psi_n(x)$. Thus by GP1 we have that there exists $c \in \mathcal{C}$ such that $D(c)\theta_{c,\alpha(c)} \models \Psi_p(x)$. It follows from LN3 that $c \in \mathcal{P}$. Hence by LP2(ii) we must have that $x = \alpha(c)$. A similar argument using GN1 and LP3 shows that there exists a clause $d \in \mathcal{N}$ such that $x = \alpha(d)$. But this contradicts the assumption that α is a clause-mapping witnessing the satisfiability of $\mathcal{C}$. This establishes that if $\mathcal{C}$ is satisfiable then $D \not\models \Phi$.

$$F(c) = \begin{array}{||cc||c||cc||cc||c||} \hline P_1^{A,O}(x,y) & & Q_1^A(y) & P_2^{A,O}(x,z) & & P_3^{A,O}(z,t) & & Q_2^A(t) \\ \hline k_1 & 2 & 2 & k_1 & o_1 & l_1 & u_1 & 3 \\ k_2 & u_1 & & k_2 & o_2 & l_2 & u_2 & \\ k_3 & u_2 & & k_3 & o_3 & l_3 & 3 & \\ \hline \end{array}$$

$$\begin{array}{rcl} dom(o_i) & = & \{l_i, 1\} \\ dom(u_i) & = & \{2, 3\} \end{array}$$

$$E_p(c) = \begin{array}{||cc||c||} \hline P_1^{A,O}(x,y) & & Q_1^A(y) \\ \hline a_1 & o_1 & 1 \\ a_2 & o_2 & \\ a_3 & o_3 & \\ \hline \end{array} \qquad E_n(c) = \begin{array}{||cc||cc||c||} \hline P_2^{A,O}(x,z) & & P_3^{A,O}(z,t) & & Q_2^A(t) \\ \hline a_1 & 4 & 4 & o_1 & 1 \\ a_2 & 5 & 5 & o_2 & \\ a_3 & 6 & 6 & o_3 & \\ \hline \end{array}$$

$$\begin{array}{rcl} \Psi_p(x) & = & \exists y[P_1^{A,O}(x,y) \wedge Q_1^A(y)] \\ \Psi_n(x) & = & \exists zt[P_2^{A,O}(x,z) \wedge P_3^{A,O}(z,t) \wedge Q_2^A(t)] \end{array}$$

Figure 6: Reduction for Φ_1

Conversely, suppose that θ is a valuation such that $D\theta \not\models \Phi$. Then for each $c \in \mathcal{C}$ we have $D(c) \not\models \Phi$. It follows by LP1 that for each $c \in \mathcal{P}$ there exists an atom $\alpha(c) \in c$ such that $D(c) \models \Psi_p(\alpha(c))$. Similarly, by LN1, for each $c \in \mathcal{N}$, there exists $\alpha(c) \in c$ such that $D(c) \models \Psi_n(\alpha(c))$. We claim that α is a clause-mapping witnessing the satisfiability of $\mathcal{C}$. For, suppose that we have $c \in \mathcal{P}$ and $d \in \mathcal{N}$ with $\alpha(c) = \alpha(d) = a$. Then $D\theta \models \Psi_p(a) \wedge \Psi_n(a)$, so that $D\theta \models \Phi$, a contradiction. This proves that if $D \not\models \Phi$ then $\mathcal{C}$ is satisfiable. ∎

C Improper Queries

The various complexity classifications of this paper have been established for proper queries only. In this appendix we briefly consider improper queries, i.e., queries which may contain repeated occurrences of a predicate. In general, our classifications of proper queries do not apply to this more general class of queries. We are not able to provide a complete classification of the complexity of improper queries, but we do establish in this section some necessary conditions and some sufficient conditions for such queries to be either in PTIME or co-NP hard.

Definition C.1 *Let V be the variables of a query Φ. A function $\alpha : V \to V$ is called an* automorphism *of Φ if $P(\alpha(v_1), \ldots, \alpha(v_k))$ is a literal of Φ whenever $P(v_1, \ldots, v_k)$ is a literal of Φ. If α sets*

$F(c) =$

$P_1^O(x)$	$P_2^{A,O}(x,y)$		$Q_1^A(y)$	$P_3^{A,O}(x,z)$		$Q_2^A(z)$
u	k_1	o_1	2	k_1	4	4
	k_2	o_2		k_2	4	
	k_3	o_3		k_3	4	

$$
\begin{aligned}
dom(o_i) &= \{1, 2\} \\
dom(u) &= \{k_1, k_2, k_3\} \\
dom(v) &= \{a_1, a_2, a_3\}
\end{aligned}
$$

$E_p(c) =$

$P_1^O(x)$
v

$E_n(c) =$

$P_2^{A,O}(x,y)$		$Q_1^A(y)$	$P_3^{A,O}(x,z)$		$Q_2^A(z)$
a_1	5	5	a_1	o_1	1
a_2	5		a_2	o_2	
a_3	5		a_3	o_3	

$$
\begin{aligned}
\Psi_p(x) &= P_1^O(x) \\
\Psi_n(x) &= \exists yz[P_2^{A,O}(x,y) \wedge Q_1^A(y) \wedge P_3^{A,O}(x,z) \wedge Q_2^A(z)]
\end{aligned}
$$

Figure 7: Reduction for Φ_2

$F(c) =$

$P_1^O(x)$	$P_2^{A,O}(x,y)$		$P_3^{A,O}(y,z)$		$Q^A(z)$
u	k_1	o_1	2	3	3
	k_2	o_2			
	k_3	o_3			

$$\begin{aligned} dom(o_i) &= \{1,2\} \\ dom(u) &= \{k_1,k_2,k_3\} \\ dom(v) &= \{a_1,a_2,a_3\} \end{aligned}$$

$E_p(c) =$

$P_1^O(x)$
v

$E_n(c) =$

$P_2^{A,O}(x,y)$		$P_3^{A,O}(y,z)$		$Q^A(z)$
a_1	4	4	o_1	1
a_2	5	5	o_2	
a_3	6	6	o_3	

$$\begin{aligned} \Psi_p(x) &= P_1^O(x) \\ \Psi_n(x) &= \exists yz[P_2^{A,O}(x,y) \wedge P_3^{A,O}(y,z) \wedge Q^A(z)] \end{aligned}$$

Figure 8: Reduction for Φ_3

$F(c) =$

$P_1^{A,O}(x,y)$		$Q_1^A(y)$	$P_2^{A,O}(x,z)$		$Q_2^A(z)$	$P_3^{A,O}(x,t)$		$Q_3^A(t)$
k_1	o_1	2	k_1	3	3	k_1	u_1	4
k_2	o_2		k_2	u_1		k_2	u_2	
k_3	o_3		k_3	u_2		k_3	4	

$$\begin{aligned} dom(o_i) &= \{1,2\} \\ dom(u_i) &= \{3,4\} \end{aligned}$$

$E_p(c) =$

$P_1^{A,O}(x,y)$		$Q_1^A(y)$	$P_2^{A,O}(x,z)$		$Q_2^A(z)$
a_1	5	5	a_1	o_1	1
a_2	5		a_2	o_2	
a_3	5		a_3	o_3	

$E_n(c) =$

$P_3^{A,O}(x,t)$		$Q_3^A(t)$
a_1	o_1	1
a_2	o_2	
a_3	o_3	

$$\begin{aligned} \Psi_p(x) &= \exists yz[P_1^{A,O}(x,y) \wedge Q_1^A(y) \wedge P_2^{A,O}(x,z) \wedge Q_2^A(z)] \\ \Psi_n(x) &= \exists t[P_3^{A,O}(x,t) \wedge Q_3^A(t)] \end{aligned}$$

Figure 9: Reduction for Φ_4

up a one-to-one correspondence between the literals of Φ then we say that it is an isomorphism. *A query Φ with the property that every automorphism is an isomorphism is called a* minimal *query.*

It can be shown that a query is minimal just when there is no proper subquery which is logically equivalent. Thus a minimal query is not logically equivalent to any query with fewer literals. Every query is equivalent to some minimal query.

Let Φ be an arbitrary query, possibly improper, and consider the literals to be ordered. We define the proper query $Diff(\Phi)$ as follows. Let $l = P(v_1, \ldots, v_k)$ be the i-th literal of Φ with predicate P. We introduce a new predicate P^i and replace l by the literal $P^i(v_1, \ldots, v_k)$. For example if Φ is the query

$$P(u,v) \wedge P(u,x) \wedge Q(v) \wedge Q(x) \wedge R(x)$$

then $Diff(\Phi)$ is the query

$$P^1(u,v) \wedge P^2(u,x) \wedge Q^1(v) \wedge Q^2(x) \wedge R(x)$$

Theorem C.1 *If Φ is a minimal query then $Diff(\Phi) \leq_I \Phi$ and $Diff(\Phi) \leq_{II} \Phi$.*

Proof: We construct a polynomial time function f mapping databases for the query $Diff(\Phi)$ to databases for the query Φ such that $D \models Diff(\Phi)$ if and only if $f(D) \models \Phi$. The domain of the database $f(D)$ will consist of pairs of the form $a.v$ where a is an element of the domain of D and v is one of the variables of Φ. If o is an OR-object of D and v is a variable of Φ then $f(D)$ will contain an OR-object $o.v$ with

$$Dom(o.v) = \{a.v | a \in Dom(o)\}.$$

For each atom $P^i(u_1, \ldots, u_k)$ of D, where the u_i are either OR-objects or constants, the database $f(D)$ will contain an atom $P(u_1.v_1, \ldots, u_k.v_k)$. It is readily checked that if D is a type I or type II database then so is $f(D)$.

Suppose that $D \models Diff(\Phi)$. Let θ' be a valuation of $f(D)$. Since D is a type II database, and the query $Diff(\Phi)$ is proper, there exists for each OR-object o of D a unique variable v such that $o.v$ is an OR-object of $f(D)$. Hence we may define the valuation θ of D by letting $\theta(o)$ be the constant a such that $\theta'(o.v) = a.v$. Since $D \models Diff(\Phi)$ there exists an assignment α of the variables of $Diff(\Phi)$ such that $D\theta \models Diff(\Phi)\alpha$. Define the assignment α' of the variables of Φ by $\alpha'(v) = \alpha(v).v$. Now for each literal $l = P^i(v_1, \ldots, v_k)$ of $Diff(\Phi)$ there exists an atom $\tau = P^i(u_1, \ldots, u_k)$ of D such that $l\alpha = \tau\theta$. The literal l of $Diff(\Phi)$ corresponds to the literal $l' = P(v_1, \ldots, v_k)$ of Φ. Similarly, the atom τ corresponds to the atom $\tau' = P(u_1.v_1, \ldots, u_k.v_k)$ of $f(D)$. Now note that we have

$$\begin{aligned} l'\alpha' &= P(u_1.v_1, \ldots, u_k.v_k) \\ &= \tau'\theta' \end{aligned}$$

This establishes that $f(D)\theta' \models \Phi$. Thus we have shown that $D \models Diff(\Phi)$ implies $f(D) \models \Phi$.

For the converse, suppose that $f(D) \models \Phi$. Let θ' be the valuation on $f(D)$ defined by $\theta'(o.v) = \theta(o).v$. Then there exists an assignment α of the variables of Φ such that $f(D)\theta' \models \Phi\alpha$. Since each element of the domain of $f(D)$ is of the form $a.v$ we may write $\alpha(w) = \alpha_1(w).\alpha_2(w)$. The function α_2 operates on the variables of Φ. For each literal $P(v_1, \ldots, v_k)$ of Φ, there exists an atom

$$P(\alpha_1(v_1).\alpha_2(v_1), \ldots, \alpha_1(v_k).\alpha_2(v_k))$$

of $f(D)\theta'$. By construction of $f(D)$, this means that there exists a literal

$$P(\alpha_2(v_1), \ldots, \alpha_2(v_k))$$

of Φ. This shows that α_2 is an automorphism of Φ. Since Φ is a minimal query, it follows that α_2 is in fact an isomorphism of Φ. In particular, it follows that α_2 is a bijection on the variables of Φ. Since Φ and $Diff(\Phi)$ have the same variables we may define the assignment β of the variables of $Diff(\Phi)$ in D by

$$\beta(w) = \alpha_1(\alpha_2^{-1}(w)).$$

We claim that $D\theta \models Diff(\Phi)\beta$. For, let $l = P^i(v_1, \ldots, v_k)$ be a literal of the query $Diff(\Phi)$. Because α_2 is an isomorphism of Φ, there exists a literal $P(w_1, \ldots, w_k)$ of Φ such that $\alpha_2(w_j) = v_j$ for each j. Let $\tau' = P(u_1.v_1, \ldots, u_k.v_k)$ be the atom of $f(D)$ with $\tau'\theta' = P(\alpha(w_1), \ldots, \alpha(w_k))$. Because the query Φ is minimal, it contains only one literal with predicate P and arguments $(v_1, \ldots, v_k)$, hence by construction of $f(D)$ the atom τ' could only have arisen from the atom $\tau = P^i(u_1, \ldots, u_k)$ of D. Now notice that

$$\begin{aligned} \theta(u_j) &= \alpha_1(w_j) \\ &= \alpha_1(\alpha_2^{-1}(v_j)) \\ &= \beta(v_j) \end{aligned}$$

This shows that $\tau\theta = l\beta$, and so $D \models Diff(\Phi)$. We have now established that $f(D) \models \Phi$ implies $D \models Diff(\Phi)$. ∎

It is a corollary of this result that if $Diff(\Phi)$ is co-NP hard then Φ is co-NP hard also. Hence the result yields a sufficient condition for hardness of improper queries. We note that the proof breaks down for the relation $Diff(\Phi) \leq_{III} \Phi$ because on type III databases we cannot associate a unique variable with OR-objects when they occur in distinct relations of $Diff(\Phi)$. However we can show the converse relation for type III databases.

Theorem C.2 *For any query Φ we have $\Phi \leq_{III} Diff(\Phi)$.*

Proof: If D is a database for the query Φ let $f(D)$ be the database for the query $Diff(\Phi)$ obtained by making each relation P^i of $f(D)$ be a copy of the relation P of D. That is, $P^i(u_1, \ldots, u_k)$ is an atom of $f(D)$ whenever $P(u_1, \ldots, u_k)$ is an atom of D. It is straightforward to show that $D \models \Phi$ if and only if $f(D) \models Diff(\Phi)$. ∎

This gives sufficient conditions for improper queries to have PTIME data complexity: if $Diff(\Phi)$ has PTIME data complexity then so has Φ. Observe that the construction does not work for type I or type II databases because it results in OR-objects which are shared amongst distinct relations. We now provide examples that indicate that the inverses of the relations of the above two results are unlikely to hold.

Example C.1 (Counter Example to the converse of Theorem C.1)

Let Φ be the query

$$P^O(x) \wedge Q^{A,O}(x, y) \wedge Q^{A,O}(y, z) \wedge R^A(z)$$

Note that $Diff(\Phi)$ is acyclic and therefore has polynomial time complexity on type II databases. However, Φ has co-NP hard data complexity on type I databases. We will not give a complete proof of this. The following table demonstrates that the subquery

$$\Phi_1(x) = \exists yz[Q^{A,O}(x, y) \wedge Q^{A,O}(y, z) \wedge R^A(z)]$$

fails to have the definite answer property. Hardness follows from this using ideas similar to those of Appendix B.

$Q^{A,O}(x,y)$		$R^A(y)$
a	b	d
b	o	
c	d	

Here the OR-object o has domain $Dom(o) = \{c, d\}$. Note that we have

$$D \models \Phi_1(a) \vee \Phi_1(b)$$

but neither of the disjuncts is entailed separately. Thus if $\Phi \leq_I Diff(\Phi)$ or $\Phi \leq_{II} Diff(\Phi)$ then P=NP, which is unlikely.

Example C.2 (Counter Example to the converse of Theorem C.2)

Let Φ be the query

$$P^O(x) \wedge Q_1^{A,O}(x, y) \wedge R^A(y) \wedge Q_2^{A,O}(x, z) \wedge R^A(z)$$

It follows from the results of Section 10 that $Diff(\Phi)$ has co-NP hard data complexity on type III databases. However, Φ has PTIME data complexity on type III databases. This follows from the fact that the subquery

$$\Phi_1(x) = \exists yz[Q_1^{A,O}(x, y) \wedge R^A(y) \wedge Q_2^{A,O}(x, z) \wedge R^A(z)]$$

has the definite answer property, which means that the algorithm of Section 6 applies. To see that the definite answer property holds, consider the valuation θ which assigns to each OR-object o an element of $Dom(o)$ outside of the relation R, if possible, and otherwise an arbitrary element of $Dom(o)$. It is straightforward to show that $D\theta \models \Phi_1(x)$ implies $D \models \Phi_1(x)$. Thus if $Diff(\Phi) \leq_{III} \Phi$ then again P=NP, hence this is unlikely to be the case.

References

[1] Abiteboul, S., Kanellakis, P., and Grahne, G. On the Representation and Querying of Sets of Possible Worlds. *Theoretical Computer Science*, 78(1):159–187, 1991.

[2] Chandra, A.K. and Harel, D. Structure and Complexity of Relational Queries. *Journal of Computer and Systems Sciences*, 25(1):99–128, 1982.

[3] Garey, M. and Johnson, D. *Computers and Intractability, A Guide to Theory of NP-Completeness*. Computer Science. W.H. Freeman and Company, 1979.

[4] Imielinski, T and Lipski, W. Incomplete Information in Relational Databases. *Journal of ACM*, 31(4):761–791, October 1984.

[5] Imielinski, T., Naqvi, S., and Vadaparty, K. Incomplete Objects – A Data Model for Design and Planning Applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 288–297, Denver, Colorado, May 1991.

[6] Imielinski, T., Naqvi, S., and Vadaparty, K. Querying Designing and Planning Databases. In *Proceedings of the Second International Conference on Deductive Object-Oriented Databases*, pages 524–545, Munich, Germany, December 1991.

[7] Imielinski, T. and Vadaparty, K. Complexity of Querying Databases with OR-Objects. In *Proceedings of the Eighth ACM Symposium on Principles of Database Systems (PODS)*, pages 51–65, 1989.

[8] Immerman, N. Relational Queries Computable in Polynomial Time. *Information and Control*, 68:86–104, 1986.

[9] Lloyd, J. *Foundations of Logic Programming*. Springer Verlag, 1984.

[10] Melton, J and Simon, A. *Understanding the New SQL: A Complete Guide*, volume 1. Morgan Kaufmann, 1993.

[11] Minker, J. and Rajasekhar, A. A Fixpoint Semantics for Disjunctive Logic Programs. *Journal of Logic Programming*, 9(1), 1990.

[12] Ullman, J.D.,. *Principles of Databases and Knowledge Base Systems, Volume I.* Computer Science Press, 1988.

[13] Van Hentenryck, P. *Constraint Satisfaction in Logic Programming*. Logic Programming. MIT Press, 1989.

[14] Vardi, M. The Complexity of Relational Query Languages. In *Proceedings of the 14th STOC*, pages 137–146, 1982.

[15] Vardi, M. Querying Logical Databases. *J. of Computer and System Sciences*, 33(2):142–160, 1986.