On the Complexity of Data Disjunctions *

Thomas Eiter
Institut für Informationssysteme
Abteilung für Wissensbasierte Systeme
eiter@kr.tuwien.at

Helmut Veith
Institut für Informationssysteme
Abteilung für Datenbanken und AI
veith@dbai.tuwien.ac.at

Technische Universität Wien
A-1040 Wien, Austria

Technical Report DBAI-TR-99-29

Abstract

We study the complexity of data disjunctions in disjunctive deductive databases (DDDBs), i.e., minimal clauses $R(\bar{c}_1) \vee \cdots \vee R(\bar{c}_k)$, $k \geq 2$, derived from the database in which all atoms involve the same predicate R. We consider deciding existence and uniqueness of a data disjunction, as well as actually computing one, both for propositional (data) and nonground (program) complexity of the database. Our results extend and complement previous results on the complexity of disjunctive databases, and provide tools for the analysis of the complexity of function computation using upgrading techniques, which we develop for this purpose.

1 Introduction

During the past decades, a lot of research has been spent to overcome the limitations of conventional relational database systems. The field of deductive databases, which has emerged from logic programming [29], uses logic as a tool for representing and querying information from databases. Numerous logical query languages, which extend Horn clause programming for dealing with various aspects such as incomplete or indefinite information, have been proposed to date, cf. [1, 33].

In particular, the use of disjunction in rule heads for expressing indefinite information was proposed in Minker's seminal paper [32], which started interest in disjunctive logic programming [30, 10]. For example, the rule

$$lives\_in(x, US) \vee lives\_in(x, canada) \vee lives\_in(x, mexico) \leftarrow lives\_in(x, n\_america) \quad (1)$$

informally states that a person living in north America lives in one of the three countries there. The semantical and computational aspects of disjunctive logic programming, and in particular, disjunctive deductive databases, have been investigated in many papers (see [33] for an overview).

*The results of this paper have been presented at the international workshop "Colloquium Logicum: Complexity," Vienna, October 9–10, 1998. This work was partially supported by the Austrian Science Fund Project N Z29-INF, and by the British Council-Austria ARC Programme for collaborative research (Oracle Computations within Descriptive Complexity Theory).

In this paper, we are interested in a restricted type of disjunction, which has been previously considered e.g. in [6, 5, 12, 19, 15]. A *data disjunction* [19] is a ground clause $R(\bar{c}_1) \vee \cdots \vee R(\bar{c}_k)$, $k \geq 2$, in which all atoms are different and involve the same predicate R. For example, the head of the rule (1) for $x = Joe$, is a data disjunction, as well as the disjunctive fact

$$loves(bill, monica) \vee loves(bill, hillary).$$

A data disjunction expresses indefinite information about the truth of a predicate on a set of arguments; in database terminology, it expresses a *null value* on this predicate, whose range is given by the arguments $\bar{c}_1, \ldots, \bar{c}_k$ of its atoms. In the context of deductive databases, null values of this form in the extensional database and their complexity have been considered e.g. in [20], and in many other papers.

If, in the above example, the fact $lives\_in(joe, n\_america)$ is known, then the data disjunction

$$C: \quad lives\_in(joe, US) \vee lives\_in(joe, canada) \vee lives\_in(joe, mexico)$$

can be derived from rule (1). If a clause C is entailed from a database, then also any clause C' subsumed by C is entailed. For example, the clause $C \vee lives\_in(joe, usbekistan)$ is entailed by virtue of C as well. We thus adopt the natural condition that a data disjunction C must be minimal, i.e., no proper subclause of C is entailed.

The question we address here is the complexity of data disjunctions in a disjunctive deductive database (DDDB). Table 1 summarizes the problems studied in this paper (see Section 3 for precise definitions), and the main complexity results obtained. They complement previous results on reasoning from DDDBs. Deciding whether an arbitrary disjunction, rather than a data disjunction, follows from a DDDB has Π_2^P data and propositional complexity, and exponentially higher expression and combined complexity [14]; various syntactic restrictions lower the complexity to coNP or even polynomial time [9]. On the other hand, evaluating a conjunctive query over a disjunctive extensional database is coNP-complete [20], and hence deciding entailment of a single ground atom a has coNP data and propositional complexity. Thus, data disjunctions have intermediate complexity between arbitrary clauses and single atoms.

Observe that Table 1 contains also results on actually computing a data disjunction (assuming at most one exists). While all the results in this table could be derived in the standard way, i.e., by proving membership in class C and reducing a chosen C-hard problem to the problem in question, we pursue here an "engineering" perspective of complexity analysis in databases, proposed e.g. in [18], which utilizes tools from descriptive and succinct complexity theory and exploits properties of the deductive database semantics. By means of these tools, hardness results can be derived at an abstracted level of consideration, without the need for choosing a fixed C-hard problem. Such tools (in particular, complexity upgrading) have been developed for decision problems, but are not available for function problems. We overcome this by generalizing the tools for propositional problems in a suitable way.

Thus, the main contributions of this paper can be summarized as follows. Firstly, we determine the complexity of data disjunctions. We obtain natural and simple logical inference problems complete for the class Θ_2^{P} of the refined polynomial hierarchy [45], and, in their computational variants, complete problems for the function classes $\mathrm{FP}_{\|}^{\mathrm{NP}}$ and $\mathrm{FL}_{\log}^{\mathrm{NP}}[\log]$ and their exponential analogs. Secondly, we provide upgrading techniques for determining the complexity of function computations. They generalize available tools for decisional problems and may be fruitfully applied in other contexts as well.

The rest of this paper is organized as follows. Section 2 states preliminaries, and Section 3 formalizes the problems. In Section 4, the decision problems are considered, while Section 5 is devoted to computing

Data Disjunction
Input: A disjunctive deductive database $\mathsf{DB} = (\pi, E)$, where E is a collection of (possibly disjunctive) ground facts, and π are the inference rules, plus a distinguished relation symbol R.

	propositional complexity (π ground)	*data complexity* (π fixed)	*expression complexity* (E fixed)	*combined complexity* (π, E free)
$\exists$DD: does DB have a data disjunction on R?				
	Θ_2^{P}	Θ_2^{P}	$\mathrm{PSpace}^{\mathrm{NP}}$	$\mathrm{PSpace}^{\mathrm{NP}}$
$\exists!$DD: does DB have a unique data disjunction on R?				
	Θ_2^{P}	Θ_2^{P}	$\mathrm{PSpace}^{\mathrm{NP}}$	$\mathrm{PSpace}^{\mathrm{NP}}$
λDD: Computation of the unique data disjunction on R.				
	$\mathrm{FP}_{\parallel}^{\mathrm{NP}}$	$\mathrm{FP}_{\parallel}^{\mathrm{NP}}$	$\mathrm{FPSpace}^{\mathrm{NP}}$	$\mathrm{FPSpace}^{\mathrm{NP}}$
k-λDD: Computation of the unique data disjunction on R, if it has at most k disjuncts (k constant).				
	$\mathrm{FL}_{\log}^{\mathrm{NP}}[\log]$	$\mathrm{FL}_{\log}^{\mathrm{NP}}[\log]$	$\mathrm{FPSpace}^{\mathrm{NP}}[\mathrm{pol}]$	$\mathrm{FPSpace}^{\mathrm{NP}}[\mathrm{pol}]$

Table 1: Complexity of Data Disjunctions.

data disjunctions. Logical characterizations of function computations are given through a generalization of the *Stewart Normal Form* (SNF) [38, 39, 17], which has been first used to characterize the class Θ_2^{P}. For deriving the expression and combined complexity of function computations, upgrading results are developed in Section 6. The final Section 7 applies the results to the area of closed-world reasoning and gives some conclusions.

2 Preliminaries

2.1 Deductive databases

For a background on disjunctive deductive databases, we refer to [30].

Syntax. A finite relational language is a tuple $\tau = (R_1, \ldots, R_n, c_1, \ldots, c_m)$ where the R_i are relation symbols (also called predicate symbols) with associated arities $a_1, \ldots, a_n$, and the c_i are constant symbols. An atom is a formula of the form $R_i(\bar{v})$, where $\bar{v}$ is a tuple of first order variables and constant symbols.

A *disjunctive datalog rule* is a clause of the form

$$a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_m, \quad 1 \leq n, 0 \leq m$$

over a finite relational language, where the a_i's are atoms forming the *head* of the clause, and the b_j's are *atoms* or inequalities of the form $u \neq v$ (where u and v are variables or constants) forming the *body* of the clause.

A *disjunctive deductive program* (short *program*) is a finite collection of disjunctive datalog rules; it is *ground*, if no variables occur in the rules.

If a predicate symbol occurs only in rule bodies, it is called an *input predicate*, otherwise it is called a *derived predicate*.

A *disjunctive deductive program with input negation* is a program where input predicates are allowed to appear negated.

A *ground fact* is a clause of the form

$$a_1$$

where a_1 is a variable-free atom; a *disjunctive ground fact* is a clause of the form

$$a_1 \vee \ldots \vee a_n$$

where the a_i's are variable-free atoms.

A *disjunctive deductive database* (DDDB) is a tuple $\mathsf{DB} = (\pi, E)$ where π is a program, and E is a finite set of disjunctive ground facts. Here, E represents the input database, also called the *extensional part*, and π are inference rules, called the *intensional part* of the database DB.

Remark: Note that π, E, and $\pi \cup E$ are all disjunctive deductive programs, i.e., ground facts can be included into the programs, and in fact we shall do this for defining the semantics. However, for methodological and complexity issues, it is important to distinguish the input data from the inference rules. For example, the complexity of evaluating DB is exponentially lower when π is fixed. In section 3, we shall define data and query complexity to give a formal meaning to this intuition.

Semantics. The semantics of DDDBs has been defined in terms of their minimal models [32, 30]. For a DDDB $\mathsf{DB} = (\pi, E)$, we denote by $\mathsf{HU}_{\mathsf{DB}}$ its Herbrand universe, i.e., the set of all constants occurring in DB. The Herbrand base $\mathsf{HB}_{\mathsf{DB}}$ (resp., *disjunctive Herbrand base* $\mathsf{DHB}_{\mathsf{DB}}$) is the set of all ground atoms (resp., disjunctive ground facts) of predicates in DB over $\mathsf{HU}_{\mathsf{DB}}$. The ground instantiation of a program π over a set of constants C is denoted by $\mathrm{ground}(\pi, C)$; the ground instance of DB, denoted $\mathrm{ground}(\mathsf{DB})$, is $\mathrm{ground}(\pi, \mathsf{HB}_{\mathsf{DB}}) \cup E$.

An (Herbrand) interpretation of DB is a subset $H \subseteq \mathsf{HB}_{\mathsf{DB}}$. An interpretation H of DB is a *model* of DB, if it satisfies each rule in $\mathrm{ground}(\mathsf{DB})$ in the standard sense. A model H of DB is minimal, if it does not contain any other model of DB properly; by $\mathrm{MM}(\mathsf{DB})$ we denote the set of all minimal models of DB. We write $\mathsf{DB} \models_{\mathrm{MM}} \varphi$ if a formula φ is true in every $M \in \mathrm{MM}(\mathsf{DB})$, and say that φ is entailed from DB.

Example 2.1 *Let* $\mathsf{DB} = (\pi, E)$*, where* π *is the rule* $q(x) \leftarrow p(x)$ *and* E *contains the single disjunctive fact* $p(a) \vee q(b)$*. Then* $M = \{p(a), q(a)\}$*,* $M' = \{p(a), q(a), q(b)\}$ *are among the models of* DB*;* M *is minimal, while* M *is not. The minimal models of* DB *are* $\mathrm{MM}(\mathsf{DB}) = \{\{p(a), q(a)\}, \{q(b)\}\}$.

Remark. It is easy to see [32] that for each positive clause C, $\mathsf{DB} \models_{\mathrm{MM}} C$ if and only iff $\mathsf{DB} \models C$, where $\models$ is satisfaction in all models of DB. We will repeatedly use this fact.

The set of minimal models of DB has been characterized in terms of a unique least model-state MS (see [30]), i.e., a subset of $\mathsf{DHB}_{\mathsf{DB}}$, which can be computed by least fixpoint iteration of an operator T_P^S generalizing the standard T_P operator of logic programming [29]. In general, the computation of MS takes exponential space and time, even if the program π of DB is fixed.

2.1.1 Negation

Introducing negation in disjunctive deductive databases is not straightforward, and gave rise to different semantics, cf. [33]. We restrict here to *input negation*, i.e., the use of negated atoms $\neg R(\bar{t})$ in rule bodies where R is an extensional predicate, and adopt a *closed-world assumption (CWA) on models* imposing the following condition: any accepted model M of $\mathsf{DB} = (\pi, E)$, restricted to the extensional part, must be a minimal model of E. Unless stated otherwise, a model of a DDDB must satisfy this kind of closed-world assumption.

Observe that this condition is satisfied by each $M \in \mathrm{MM}(\mathsf{DB})$ if π is negation-free; furthermore, if E contains no disjunctive facts, then $\neg R(\bar{c})$ is true in every $M \in \mathrm{MM}(\mathsf{DB})$ iff $R(\bar{c}) \notin E$.

As for complexity, it is easy to see that checking whether the restriction of M to its extensional part is a minimal model of E is possible in polynomial time. Hence, the complexity of model checking and of deciding $\mathsf{DB} \models_{\mathrm{MM}} \varphi$ does not increase through the CWA on models. Furthermore, if E is restricted to disjunction-free ground facts, input negation can be eliminated in computation as follows.

Definition 2.1 *Let τ be a finite relational language, and let $\tau' = \tau \cup \{R' \mid R \in \tau\}$. Then $\mathrm{NEG}_{\tau'}$ denotes the class of all finite τ' structures $\mathcal{A}$ where for all relations R in τ, $R'^{\mathcal{A}}$ is the complement of $R^{\mathcal{A}}$.*

Proposition 2.1 *Extending a given τ-structure to its corresponding $\mathrm{NEG}_{\tau'}$ structure and replacing literals $\neg R(\bar{t})$ in a program π by $R'(\bar{t})$ is possible in LOGSPACE.*

In the derivation of hardness results, we shall consider DDDBs $\mathsf{DB} = (\pi, E)$ using input negation but where E is disjunction-free. Hence, all hardness results in this paper hold for DDDBs without negation and non-disjunctive (i.e., relational) facts as well.

2.2 Complexity

In this section, we introduce some of the more specific complexity classes and notions employed in the paper; we assume however some familiarity with basic notions of complexity theory such as oracle computations, NP, PSpace, L etc.

The class Θ_2^{P} contains the languages which are polynomial-time truth-table reducible to sets in NP. It has a wide range of different characterizations [45, 21]. In particular, the following classes coincide with Θ_2^{P}:

- $\mathrm{P}^{\mathrm{NP}}_{\|[k]}$: polynomial time computation with k rounds of parallel queries to an NP oracle [27].
- $\mathrm{P}^{\mathrm{NP}}_{\log}$: polynomial time computation where the number of queries to an NP oracle is at most logarithmic in the input size [25].
- $\mathrm{L}^{\mathrm{NP}}_{\log}$: logarithmic space computation where the number of queries to an NP oracle is at most logarithmic in the input size [26].[1]

[1] Observe that the space for the oracle tape is not bounded. Unbounded oracle space is also assumed for all other classes using an oracle in this paper.

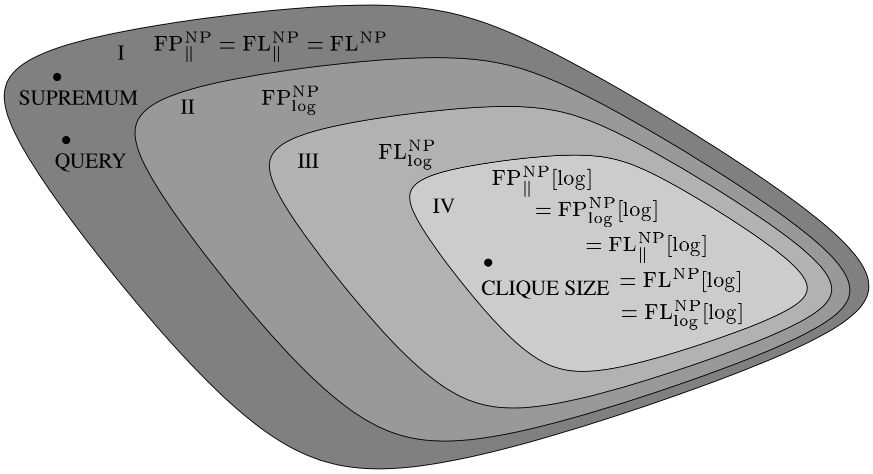

Figure 1: Function classes corresponding to Θ^{P}_2.

For an overview of different characterizations and their history, consult [45, 21]. It is shown in [21, 4, 37, 40] that this picture changes when we turn to function computation. The above mentioned list gives rise to at most three presumably different complexity classes $\mathrm{FP}^{\mathrm{NP}}_{\|}$, $\mathrm{FP}^{\mathrm{NP}}_{\log}$, and $\mathrm{FL}^{\mathrm{NP}}_{\log}$, which are shown in Figure 1. Here, for any function class FC, we denote by $\mathrm{FC}[\log]$ the restriction of FC to functions with logarithmic output size. Moreover, $\|[k]$ denotes k rounds of parallel queries, where k is a constant.

The relationships between the complexity classes in Figure 1 have been attracting quite some research efforts, which led to a number of interesting results.

- II=III is equivalent to $\mathrm{P} = \mathrm{L}$ [21].
- I=II is equivalent to the property that SAT is $O(\log n)$ approximable. This was shown in [2], after I⇒II was proved in [8]. (Here *f-approximability* of a set A means that there is a function g such that for all $x_1, \dots, x_m$ where $m \geq f(\max_i |x_i|)$ it holds that $g(x_1, \dots, x_m) \in \Sigma^m$ and $g(x_1, \dots, x_m) \neq \chi_A(x_1, \dots, x_m)$.)
- Furthermore, if I=II, then (1SAT,SAT), i.e., promise SAT, is in P [4, 40], FewP=P, NP=R [37], $\mathrm{coNP} = \mathrm{US}$, SAT $\in \mathrm{NP}(n/\log^k n)$, and $\mathrm{NP} \subseteq$ DTIME($2^{n^{O(1/\log\log n)}}$) [21].

To compare the complexity of functions, and to obtain a notion of completeness in function classes, we use Krentel's notion of metric reducibility [25]:

Definition 2.2 *A function f is* **metric reducible** *($\leq^{mr}$-reducible) to a function g (in symbols, $f \leq^{mr} g$), if there is a pair of polynomial-time computable functions h_1 and h_2 such that for every x, $f(x) = h_2(x, g(h_1, x)))$.*

Proviso 1. Let C be a complexity class. Unless stated otherwise, we use the following convention: C-completeness is defined with respect to LOGSPACE reductions, if C is a class of decision problems, and with respect to metric reductions, if C is a class of function problems.

Some complete problems for function classes are shown in Figure 1. The canonical $\mathrm{FP}_{\|}^{\mathrm{NP}}$-complete problem is QUERY, i.e., computing the string $\chi(I_1)\chi(I_2)\cdots\chi(I_n)$ of given SAT instances $I_1,\ldots,I_n$; SUPREMUM is computing, given a Boolean formula $F(x_1,\ldots,x_n)$, the string $s_1\ldots s_n$ where $s_i = 1$ if there is a satisfying assignment to the variables of F such that $x_i = 1$, and $s_i = 0$ otherwise; CLIQUE SIZE is computing the size of a maximum clique in a given graph. Note that this problem is also complete for $\mathrm{FP}_{\log}^{\mathrm{NP}}$. All these problems, turned into proper decision problems, are Θ_2^{P}-complete. In particular, deciding whether the maximum clique size in a graph is even and deciding whether the answer string to QUERY contains an even number of 1's are Θ_2^{P}-complete, cf. [45].

2.3 Queries and descriptive complexity

Definition 2.3 *Let τ be a finite relational language, and let $\delta = \{R\}$ be a language containing a single relational symbol R. A* query *Q is a function which maps τ-structures to δ-structures over the same domain, s.t. $Q(\mathcal{A})$ and $Q(\mathcal{B})$ are isomorphic, if $\mathcal{A}$ and $\mathcal{B}$ are isomorphic. If R is nullary, then Q is a Boolean query.*

A Boolean query Q is regarded as a mapping from τ-structures to $\{0,1\}$ s.t. for isomorphic $\mathcal{A},\mathcal{B}$, $Q(\mathcal{A}) = Q(\mathcal{B})$. If $Q(\mathcal{A}) = 1$, we also write $\mathcal{A} \models Q$.

Remarks. (1) If we disregard nonelementary queries, we can identify queries with higher order definable relations. (2) Note that "query" is also used for oracle calls. (3) Since queries are functions, we shall also write them as sets of pairs $(\mathcal{A}, Q(\mathcal{A}))$.

Definition 2.4 *Let τ be a finite relational language with a distinguished binary relation* succ*, and two constant symbols* $\min, \max$*. Then SUCC_τ is the set of all finite structures $\mathcal{A}$ with at least two distinct elements where $\mathrm{succ}^{\mathcal{A}}$ is a successor relation on $|\mathcal{A}|$, and $\min^{\mathcal{A}}, \max^{\mathcal{A}}$ are the first and last element wrt the successor relation, respectively.*

Note that queries are not defined over SUCC_τ, but over arbitrary τ-structures; this is called "order independence" of queries. Many query languages however seem to require a built-in order for capturing complexity classes, i.e., capturing requires that the τ-structures are extended by a contingent ordering to structures from SUCC_τ. Thus, when we write *on ordered structures/databases*, or on SUCC_τ, we mean that the queries are computed on τ-structures which are extended to SUCC_τ structures.

The following theorems provide examples of this phenomenon.

Definition 2.5 *A SNF formula (*Stewart Normal Form*) is a second-order formula of the form*

$$\exists\bar{x}.\alpha(\bar{x},\bar{y}) \wedge \neg\beta(\bar{x},\bar{y})$$

where α and β are Π_1^1 second order formulas with equality having the free variables $\bar{y}$. An SNF sentence is a SNF formula without free variables. The Skolem functions for the variables $\bar{x}$ are called SNF witnesses.

Lemma 2.2 ([38, 39, 17]) *Every Θ_2^{P}-computable property on SUCC_τ is expressible as*

$$\{\mathcal{A} \in \mathrm{SUCC}_\tau : \mathcal{A} \models \sigma\}$$

where σ is a SNF sentence.

This result, in equivalent terms of first-order logic with NP-computable generalized quantifiers, is contained for particular cases of generalized quantifiers in [38, 39], and was given for broad classes of generalized quantifiers in [17].

On a structure $\mathcal{A}$, a formula $\varphi(\bar{x})$ with free variables $\bar{x}$ defines the relation $\varphi^{\mathcal{A}}$ given by $\{\bar{c} \mid \mathcal{A} \models \varphi(\bar{c})\}$. A program π defines a relation R on $\mathcal{A}$, if $(\pi, \mathcal{A}) \models_{\mathrm{MM}} R(\bar{c})$ iff $\bar{c} \in R$, for every $\bar{c}$ on $\mathcal{A}$. In particular, if R is nullary, φ (resp., π) defines a property on $\mathcal{A}$.

Lemma 2.3 (immediate from [13, 14]) *Every Π_1^1 definable property φ on SUCC_τ is expressible by a disjunctive datalog program π_φ using input negation.*

Remark: Note that Lemma 2.3 does not require inequalities in rule bodies, since inequality is definable in the presence of order, cf. [14].

3 Data Disjunctions

Definition 3.1 *Given a DDDB* DB*, a disjunctive ground fact $R\bar{c}_1 \vee \ldots \vee R\bar{c}_n$, $n \geq 2$, is called a* data disjunction*, if*

1. $\mathsf{DB} \models_{\mathrm{MM}} R\bar{c}_1 \vee \ldots \vee R\bar{c}_n$*, and*
2. *for all $S \subset \{1, \ldots, n\}$,* $\mathsf{DB} \not\models_{\mathrm{MM}} \bigvee_{i \in S} R\bar{c}_i$.

In this case, we say that DB *has a data disjunction on R.*

A data disjunction can be seen as a kind of null value in a data base.

Example 3.1 *The DDDB* $\mathsf{DB} = (\pi, E)$

$$\pi: \; Pc \vee Qx, \quad Pa \vee Py \leftarrow Qy, Rxy, \quad E: \; Rab, \quad Sa \vee Rbc$$

has a data disjunction $Pa \vee Pb \vee Pc$.

Definition 3.2 *Given a DDDB* DB*, the* maximal disjunction on R *(in symbols,* $\mathrm{md}(\mathsf{DB}, R)$*) is the disjunctive ground fact*

$$\bigvee_{R\bar{c} \in \mathsf{HB}_{\mathsf{DB}} : \mathsf{DB} \not\models_{\mathrm{MM}} R\bar{c}} R\bar{c}.$$

Lemma 3.1 DB *has a data disjunction on R if and only if* $\mathsf{DB} \models_{\mathrm{MM}} \mathrm{md}(\mathsf{DB}, R)$.

Proof. If DB has a data disjunction δ on R, then no atom $R\bar{c}$ of δ is implied by DB. Therefore, δ is a subclause of $\mathrm{md}(\mathsf{DB}, R)$, and thus $\mathsf{DB} \models \mathrm{md}(\mathsf{DB}, R)$. On the other hand, if $\mathsf{DB} \models_{\mathrm{MM}} \mathrm{md}(\mathsf{DB}, R)$ then clearly $\mathrm{md}(\mathsf{DB}, R)$ is not empty. Either $\mathrm{md}(\mathsf{DB}, R)$ is a data disjunction itself, or atoms of $\mathrm{md}(\mathsf{DB}, R)$

can be removed until a minimal disjunction δ^* is reached such that $\mathsf{DB} \models_{\mathrm{MM}} \delta^*$. Since by definition no atomic subformula of $\mathrm{md}(\mathsf{DB}, R)$ is implied by DB, δ^* must contain at least two different atoms. ◀

In measuring the complexity of data disjunctions, we distinguish several cases following Vardi's [41] distinction between data complexity, expression complexity (alias program complexity), and combined complexity.

Definition 3.3 *The problems ∃DD, ∃!DD, λDD, and k-λDD are defined as follows:*

Instance: *A DDDB* $\mathsf{DB} = (\pi, E)$*, and a relation symbol* R*.*

Query:
- *∃DD:* *Does* DB *have a data disjunction on* R *?*
- *∃!DD:* *Does* DB *have a unique data disjunction on* R *?*
- *λDD:* *Compute the unique data disjunction on* R *if it exists, and # otherwise.*
- *k-λDD:* *Compute the unique data disjunction on* R*, if it exists and has at most* k *disjuncts, and # otherwise.*

Observe that ∃DD, called *ignorance test* in [5], has been used in [5, 6] to discriminate the expressive power of different query languages based on nonmonotonic logics over sets of disjunctive ground facts. Problem ∃!DD corresponds to the unique satisfiability problem. The uniqueness variant of a problem has often different complexity.

Definition 3.4 *Let* Π *be one of ∃DD, ∃!DD, λDD, or k-λDD.*

- *The* data complexity *of* Π *is the complexity of* Π *with parameter* π *fixed.*
- *The* expression complexity *of* Π *is the complexity of* Π *with parameter* E *fixed.*
- *The* propositional complexity *of* Π *is the complexity of* Π *where* π *is ground.*
- *The (unconstrained) complexity of* Π *is also called the* combined complexity *of* Π.

Problem Π *has combined (or propositional) complexity* C*, if* Π *is* C*-complete with respect to combined (resp. propositional) complexity.* Π *has data (or expression) complexity* C*, if* Π *is in* C *with respect to data (resp. expression) complexity for all choices of the parameter, and* Π *is* C*-complete with respect to data (resp. expression) complexity for a particular choice of the parameter.*

4 Existence of Data Disjunctions

Theorem 4.1 *Let* Q *be a fixed Boolean query. Over ordered databases, the following are equivalent:*

1. Q *is* Θ_2^{P}*-computable.*
2. Q *is definable by a SNF sentence.*

3. *There exist a program π and a relation symbol R s.t. $(\pi, \mathcal{A})$ has a data disjunction over R iff $\mathcal{A} \models Q$.*

4. *Q is equivalent to a SNF sentence whose SNF witnesses are* uniquely *defined.*

5. *There exist a program π and a relation symbol R s.t. $(\pi, \mathcal{A})$ has a* unique *data disjunction over R iff $\mathcal{A} \models Q$.*

Proof. The equivalence of 1. and 2. is stated in [17]. A close inspection of the proof in [17] shows that in fact 1. is also equivalent to 4.

3. $\rightarrow$ 1.: By Lemma 3.1, the following algorithm determines if DB has a data disjunction on R:

Algorithm DDExistence(DB, R)
1: $M := \emptyset$;
2: `for all` $R\bar{c} \in \mathsf{HB}_{\mathsf{DB}}$
3: `if not` ($\mathsf{DB} \models_{\mathrm{MM}} R\bar{c}$) `then` $M := M \cup \{R\bar{c}\}$;
4: $\varphi := \bigvee_{R\bar{c} \in M} R\bar{c}$
5: `if` $\mathsf{DB} \models_{\mathrm{MM}} \varphi$ `then return true else return false`;

Note that in line 4, φ equals $\mathrm{md}(\mathsf{DB}, R)$. The algorithm **DDExistence** works in polynomial time and makes two rounds of parallel queries to an NP oracle, and thus the problem is in $\mathrm{P}^{\mathrm{NP}}_{\|[2]} = \Theta^{\mathrm{P}}_{2}$.

5. $\rightarrow$ 1.: The following algorithm **DDUniqueness** is an extension of **DDExistence**.

Algorithm DDUniqueness(DB, R)
01: $M := \emptyset$;
02: `for all` $R\bar{c} \in \mathsf{HB}_{\mathsf{DB}}$
03: `if not` ($\mathsf{DB} \models_{\mathrm{MM}} R\bar{c}$) `then` $M := M \cup \{R\bar{c}\}$;
04: $\varphi := \bigvee_{R\bar{c} \in M} R\bar{c}$
05: `if not` ($\mathsf{DB} \models_{\mathrm{MM}} \varphi$) `then return` $\#$;
06: $N := \emptyset$;
07: `for all` $R\bar{c} \in M$
08: `if not` ($\mathsf{DB} \models_{\mathrm{MM}} \varphi \wedge \neg R\bar{c}$) `then` $N := N \cup \{R\bar{c}\}$;
09: $\psi := \bigvee_{R\bar{c} \in N} R\bar{c}$;
10: `if` $\mathsf{DB} \models_{\mathrm{MM}} \psi$ `then return true else return false`;

Note that lines 1 to 4 coincide with **DDExistence**. On line 5, the algorithm terminates if no data disjunction exists. Otherwise, all possible data disjunctions are subclauses of $\varphi = \mathrm{md}(\mathsf{DB}, R)$. Lines 6 to 9 construct a subclause ψ; it contains all those literals $R\bar{c}$ of $\mathrm{md}(\mathsf{DB}, R)$ in N which cannot be removed from $\mathrm{md}(\mathsf{DB}, R)$ without destroying the data disjunction, i.e., it contains those literals which necessarily appear in every data disjunction. Thus, if φ is a data disjunction, it is the unique one. On the other hand, if a unique data disjunction exists, it is by construction equal to φ.

Like the algorithm **DDExistence**, this algorithm also works in polynomial time making a constant number of rounds of parallel queries to an NP oracle. Hence, the problem is in Θ^{P}_{2}.

2. $\rightarrow$ 3.: Let φ be a formula of the form

$$\exists \bar{x}.\alpha(\bar{x}) \wedge \neg\beta(\bar{x}).$$

By Lemma 2.3 there exist programs π_A and π_B containing predicate symbols A and B such that for all $A(\bar{c}) \in \mathsf{HB}_{\pi_A}$ and $B(\bar{c}) \in \mathsf{HB}_{\pi_B}$,

$$(\pi_A, \mathcal{A}) \models_{\mathrm{MM}} A(\bar{c}) \text{ iff } \mathcal{A} \models \alpha(\bar{c}) \quad \text{and} \quad (\pi_B, \mathcal{A}) \models_{\mathrm{MM}} B(\bar{c}) \text{ iff } \mathcal{A} \models \beta(\bar{c}).$$

Let π be the program $\pi_A \cup \pi_B$ with the additional rules

$$\begin{array}{rcl} P(\bar{x}, \min) \vee P(\bar{x}, \max) & \leftarrow & A(\bar{x}), \\ P(\bar{x}, \min) & \leftarrow & B(\bar{x}). \end{array}$$

Observe that P does not occur in $\pi_A \cup \pi_B$, and thus, by well-known modularity properties [14, Section 5], the minimal models of π on $\mathcal{A}$ are obtained by extending the minimal models of $\pi_A \cup \pi_B$ on $\mathcal{A}$.

It is easy to see that $\mathsf{DB} = (\mathcal{A}, \pi)$ has a data disjunction on P if and only if there exists a tuple $\bar{c}$ on $\mathcal{A}$ such that $\mathcal{A} \models \alpha(\bar{c}) \wedge \neg\beta(\bar{c})$. Thus, π indeed computes property φ on SUCC_τ.

$4. \rightarrow 5.$: From the equivalence of 2. and 4., it follows that the data disjunction of program π in the proof of 2. $\rightarrow$ 3. is unique. ◀

Corollary 4.2 *The propositional and data complexity of* $\exists$*DD and* $\exists!$*DD are in* Θ_2^{P}. *The expression and combined complexity of* $\exists$*DD and* $\exists!$*DD are in* $\mathrm{PSpace}^{\mathrm{NP}}$.

Proof. It remains to consider expression and combined complexity. When the program is not fixed, the size of $\mathsf{HB}_{\mathsf{DB}}$ is single exponential in the input, and thus the algorithm **DDExistence** takes exponentially more steps. Thus, the problem is in $\mathrm{EXPTIME}_{||}^{\mathrm{NP}}$, which coincides with $\mathrm{PSpace}^{\mathrm{NP}}$ [18]. ◀

Since Θ_2^{P} has complete problems, we obtain from Theorem 4.1 the following.

Corollary 4.3 *There is a program* π *for which* $\exists$*DD and* $\exists!$*DD are* Θ_2^{P}*-hard.*

Hence, we obtain the announced result.

Theorem 4.4 *The data complexity and propositional complexity of* $\exists$*DD and* $\exists!$*DD is* Θ_2^{P}.

Note that the propositional complexity of $\exists$DD has been stated in [12]. The hardness proof there, given by a standard reduction, is far more involved; this indicates the elegance of using the descriptional complexity approach.

Since the data complexity of a query language is uniquely determined by its expressive power, two languages with the same expressive power will always have the same data complexity. Hence, data complexity is a property of semantics. Expression and combined complexity, however, *depend on the syntax of the language*. Therefore, it is in general not possible to determine the expression complexity of a query language $\mathcal{L}$ from its expressive power. Indeed, both the syntax and the semantics of $\mathcal{L}$ impact on its expression complexity. In spite of these principal obstacles, the *typical behavior* of expression complexity was often found to respect the following pattern: *If* $\mathcal{L}$ *captures* C, *then the expression complexity of* $\mathcal{L}$ *is hard for a complexity class exponentially harder than* C.

The main result of [18] shows that all query languages satisfying simple closure properties indeed match the above observation. Suppose that in a database, domain elements are replaced by tuples of domain elements. This operation is natural when a database is redesigned; for instance, entries like "John Smith" in a database $\mathcal{A}$ can be replaced by tuples ("John","Smith") in a database $\mathcal{B}$. It is natural to expect that a query $Q_{\mathcal{A}}$ over $\mathcal{A}$ can be easily rewritten into an equivalent query $Q_{\mathcal{B}}$ over $\mathcal{B}$. We call $Q_{\mathcal{B}}$ a *vectorized variant* of $Q_{\mathcal{A}}$. This is the essence of the first closure property:

Vector Closure: A query language is *uniformly vector closed*, if the vectorized variants of query expressions can be computed in LOGSPACE.

The second closure property is similar. Suppose again that a database $\mathcal{A}$ is replaced by a database $\mathcal{B}$ in such a way that all relations of $\mathcal{A}$ can be defined by views which use only unions and intersections of relations in $\mathcal{B}$. Then, it is again natural to expect that a query $Q_{\mathcal{A}}$ over $\mathcal{A}$ can be translated into an equivalent query $Q_{\mathcal{B}}$ over $\mathcal{B}$. In this case, we call $Q_{\mathcal{B}}$ an *interpretational variant* of $Q_{\mathcal{A}}$.

Interpretation Closure: A query language is *uniformly interpretation closed*, if the interpretational variants of queries can be computed from the database schemata in LOGSPACE.

In conclusion, we have the following closure condition (see [18] for a formal definition).

Definition 4.1 *A query language is* uniformly closed*, if it is uniformly vector closed and uniformly interpretation closed.*

Lemma 4.5 ([18]) *The language of DDDBs is uniformly closed.*

Combining Corollary 4.2 and the following Proposition 4.6, we obtain the expression and combined complexity of $\exists$DD and $\exists!$DD.

Proposition 4.6 ([18]) *If a language is uniformly closed, and expresses all Θ_2^{P} properties of SUCC_τ, then its expression complexity and combined complexity are at least $\mathrm{PSpace}^{\mathrm{NP}}$.*

Theorem 4.7 *The expression complexity and the combined complexity of $\exists$DD and $\exists!$DD is $\mathrm{PSpace}^{\mathrm{NP}}$.*

5 Computation of Data Disjunctions

5.1 Data complexity and propositional complexity

Theorem 5.1 *Let Q be a fixed query. Then, over ordered databases the following are equivalent.*

1. *Q is $\mathrm{FP}_{||}^{\mathrm{NP}}$ computable.*
2. *Q is definable by a SNF formula.*
3. *There exist a program π and a relation symbol R s.t. $Q(\mathcal{A})$ is polynomial time computable from the unique data disjunction of $(\mathcal{A}, \pi)$ on R.*

Proof. 1. $\to$ 2.: The problem of deciding whether a given tuple $\bar{c}$ on $\mathcal{A}$ fulfills $\bar{c} \in Q(\mathcal{A})$ is easily seen to be in Θ_2^{P}. Thus, Lemma 2.2 implies there is a SNF sentence σ such that $\mathcal{A}, \bar{c} \models \sigma$ iff $\bar{c} \in Q(\mathcal{A})$ (provide $\bar{c}$ through a designated singleton relation $R_{\bar{c}}$, and use $\exists \bar{y} R_{\bar{c}}(\bar{y})$ to access $\bar{c}$). Hence, there is an open SNF formula $\varphi(\bar{y})$ s.t. $\mathcal{A}, \bar{c} \models \varphi(\bar{y})$ iff $\bar{c} \in \pi(\mathcal{A})$.

2. $\to$ 3.: Similarly as in the proof of Theorem 4.1, let φ be the SNF formula

$$\exists \bar{x}.\alpha(\bar{x}, \bar{y}) \wedge \neg\beta(\bar{x}, \bar{y})$$

having the free variables $\bar{y}$. By Lemma 2.3 there exist programs π_A and π_B containing predicate symbols A and B such that for all $A(\bar{c}, \bar{d}) \in \mathsf{HB}_{\pi_A}$ and $B(\bar{c}, \bar{d}) \in \mathsf{HB}_{\pi_B}$

$$(\mathcal{A}, \pi_A) \models_{\mathrm{MM}} A(\bar{c}, \bar{d}) \text{ iff } \mathcal{A} \models \alpha(\bar{c}, \bar{d}) \quad \text{and} \quad (\mathcal{A}, \pi_B) \models_{\mathrm{MM}} B(\bar{c}, \bar{d}) \text{ iff } \mathcal{A} \models \beta(\bar{c}, \bar{d}).$$

We have to construct a program π whose unique data disjunction on input $\mathcal{A}$ over relation R contains the information about all tuples in $\varphi^{\mathcal{A}}$. To this end, consider the program in Figure 2. Let a be the arity of A and B there. Then the new relation symbols T and S also have arity a, and R has arity $a+2$. The program

$$\begin{aligned}
T(\overline{\min}) & & & \qquad (1)\\
S(\bar{u}) \vee T(\bar{v}) &\leftarrow & T(\bar{u}) \wedge \operatorname{succ}(\bar{u}, \bar{v}) & \qquad (2)\\
S(\overline{\max}) &\leftarrow & T(\overline{\max}) & \qquad (3)\\
R(\bar{x}, \bar{y}, \min, \min) \vee R(\bar{x}, \bar{y}, \min, \max) \vee R(\bar{x}, \bar{y}, \max, \max) &\leftarrow & S(\bar{x}\bar{y}) & \qquad (4)\\
R(\bar{x}, \bar{y}, \min, \min) \vee R(\bar{x}, \bar{y}, \min, \max) &\leftarrow & S(\bar{x}\bar{y}) \wedge A(\bar{x}\bar{y}) & \qquad (5)\\
R(\bar{x}, \bar{y}, \min, \min) &\leftarrow & S(\bar{x}\bar{y}) \wedge B(\bar{x}\bar{y}) & \qquad (6)
\end{aligned}$$

Figure 2: DDDB program for $\mathrm{FP}_{\|}^{\mathrm{NP}}$ queries.

requires that a successor relation over tuples is available. The lexicographical successor relation can be easily defined using datalog rules. Lines 1 to 3 enforce that $S(\bar{c})$ holds for at least one $\bar{c}$. Consequently, the unique data disjunction on $\bar{c}$ is the positive clause containing all possible $S(\bar{c})$ on $\mathcal{A}$.

Consider lines 1 to 4 now. If the program contained only these rules, then line 4 would enforce that the unique data disjunction on R would be the clause

$$\bigvee_{\bar{c}, \bar{d}} R(\bar{c}, \bar{d}, \min, \min) \vee R(\bar{c}, \bar{d}, \min, \max) \vee R(\bar{c}, \bar{d}, \max, \max).$$

Lines 5 and 6, however, remove certain literals from this clause. In particular, it holds that $\mathcal{A} \models \varphi(\bar{d})$ iff there is a $\bar{c}$ on $\mathcal{A}$ such that $\mathcal{A} \models \alpha(\bar{c}, \bar{d}) \wedge \neg\beta(\bar{c}, \bar{d})$ iff there is a $\bar{c}$ on $\mathcal{A}$ such that the (unique) data disjunction of $(\pi, \mathcal{A})$ on R contains the clause $R(\bar{c}, \bar{d}, \min, \min) \vee R(\bar{c}, \bar{d}, \min, \max)$ but not $R(\bar{c}, \bar{d}, \max, \max)$. Therefore, $\varphi^{\mathcal{A}}$ is polynomial time computable from the unique data disjunction on R.

3. $\to$ 1.: The algorithm **DDUniqueness** in the proof of Theorem 4.1 computes the unique data disjunction in its variable ψ, provided it exists. It is easily modified to output # in the other case. ◀

The data and propositional complexity of λDD is an easy corollary to this result.

Corollary 5.2 *Problem λDD has data complexity and propositional complexity* $\mathrm{FP}_{\|}^{\mathrm{NP}}$.

Definition 5.1 *A* domain element query *(DEQ) is a query whose answer relation is a singleton, i.e., a query Q s.t. for all $\mathcal{A}$ it holds that $|Q(\mathcal{A})| = 1$.*

Theorem 5.3 *Let Q be a fixed DEQ. Then, over ordered databases the following are equivalent:*

1. *Q is $\mathrm{FP}_{\|}^{\mathrm{NP}}[\log]$ computable.*
2. *$Q(\mathcal{A})$ is a tuple of SNF witnesses to a SNF sentence.*
3. *There exist a program π and a relation symbol R s.t. $Q(\mathcal{A})$ is definable as a projection of the unique data disjunction over R, where the data disjunction contains at most two atoms.*
4. *There exist a program π and a relation symbol R s.t. $Q(\mathcal{A})$ is polynomial time computable from the unique data disjunction over R where the data disjunction contains at most two atoms.*

Proof. $1. \to 2.$: For constant $\bar{c}$, the problem of deciding whether $\bar{c} \in \pi(\mathcal{A})$ is easily seen to be in Θ_2^{P}. Thus, there is a SNF formula $\varphi(\bar{x})$ having the free variables $\bar{x}$ for the constants s.t. $\mathcal{A}, \bar{c} \models \varphi(\bar{x})$ iff $\bar{c} \in \pi(\mathcal{A})$. By Theorem 4.1 we may w.l.o.g. suppose that the SNF witnesses in φ are unique.

$2. \to 3.$: Let φ be a SNF sentence

$$\exists \bar{x}.\alpha(\bar{x}) \wedge \neg\beta(\bar{x})$$

which has unique SNF witnesses. Then, the program for φ in part $2. \to 3.$ of the proof of Theorem 4.1 has a unique data disjunction of the form $P(\bar{c}, \bar{d}, \min) \vee P(\bar{c}, \bar{d}, \max)$. From this data disjunction, the projection on the $\bar{d}$ tuple yields the desired result.

$3. \to 4.$: Trivial.

$4. \to 1.$: Let M be the polynomial time Turing Machine which computes the answer from the unique data disjunction. Then, we use the algorithm **DDUniqueness** in the proof of Theorem 4.1 which computes the unique data disjunction in its variable ψ. It remains to check if the data disjunction is small, and to simulate M. By assumption, the result has logarithmic size.

◀

The data and propositional complexity of k-λDD is an immediate corollary to this result.

Corollary 5.4 *The data complexity and propositional complexity of k-λDD are* $\mathrm{FL}_{\log}^{\mathrm{NP}}[\log]$.

At this point, the question arises whether we could not have surpassed the reduction in the proof of $2. \to 3.$ in Theorem 5.1, by exploiting the completeness result on k-λDD. The next result tells us that this is (presumably) not possible, and that disjunctions not bounded by a constant are needed in general to have hardness for $\mathrm{FP}_{\|}^{\mathrm{NP}}$.

Proposition 5.5 *Problem λDD is metric reducible to k-λDD with respect to data complexity, for some $k \geq 2$, if and only if* $\mathrm{FP}_{\|}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$.

Proof. $\rightarrow$: Suppose λDD is metric reducible to k-λDD. Then, Corollary 5.2 implies that k-λDD is metric complete for $\mathrm{FP}_{\|}^{\mathrm{NP}}$. Since $\mathrm{FL}_{\log}^{\mathrm{NP}}[\log] \subseteq \mathrm{FP}_{\log}^{\mathrm{NP}}$, this implies that $\mathrm{FP}_{\|}^{\mathrm{NP}} \subseteq \leq^{mr}(\mathrm{FP}_{\log}^{\mathrm{NP}})$ where $\leq^{mr}(\mathbf{C})$ denotes the closure of $\mathbf{C}$ under metric reductions. Clearly, $\leq^{mr}(\mathrm{FP}_{\log}^{\mathrm{NP}}) = \mathrm{FP}_{\log}^{\mathrm{NP}}$, and thus $\mathrm{FP}_{\|}^{\mathrm{NP}} \subseteq \mathrm{FP}_{\log}^{\mathrm{NP}}$ holds. Combined with $\mathrm{FP}_{\log}^{\mathrm{NP}} \subseteq \mathrm{FP}_{\|}^{\mathrm{NP}}$ (cf. Figure 1), it follows that $\mathrm{FP}_{\|}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$.

$\leftarrow$: Suppose that $\mathrm{FP}_{\log}^{\mathrm{NP}} = \mathrm{FP}_{\|}^{\mathrm{NP}}$. Let f be any function complete for $\mathrm{FP}_{\log}^{\mathrm{NP}}$ (such an f exists). Then, $\lambda\mathrm{DD} \leq^{mr} f$ by hypothesis. We use the following fact: Every function f in $\mathrm{FP}_{\log}^{\mathrm{NP}}$ is $\leq^{mr}$-reducible to some function g in $\mathrm{FL}_{\log}^{\mathrm{NP}}[\log]$. Indeed, Krentel showed that his class $\mathrm{OptP}[\mathrm{O}(\log \mathrm{n})]$ satisfies $\mathrm{FP}_{\log}^{\mathrm{NP}} \subseteq \leq^{mr}(\mathrm{OptP}[\mathrm{O}(\log \mathrm{n})])$, and that CLIQUE SIZE (cf. Section 2) is $\mathrm{OptP}[\mathrm{O}(\log \mathrm{n})]$-complete [25]. Since CLIQUE SIZE is clearly in $\mathrm{FP}_{\|}^{\mathrm{NP}}[\log] = \mathrm{FL}_{\log}^{\mathrm{NP}}[\log]$, the claimed fact follows by transitivity of $\leq^{mr}$. This fact and Corollary 5.2, together with the hypothesis $\mathrm{FP}_{\log}^{\mathrm{NP}} = \mathrm{FP}_{\|}^{\mathrm{NP}}$ imply that $\lambda\mathrm{DD} \leq^{mr} f \leq^{mr} g \leq^{mr} k$-$\lambda$DD. By transitivity of $\leq^{mr}$, we obtain $\lambda\mathrm{DD} \leq^{mr} k$-$\lambda$DD. ◀

5.2 Expression and combined complexity

Finally, we determine the expression complexity of computing the unique data disjunction.

Theorem 5.6 *The expression and combined complexity of λDD is* $\mathrm{FPSpace}^{\mathrm{NP}}$*, and the expression and combined complexity of k-λDD is* $\mathrm{FPSpace}^{\mathrm{NP}}[\mathrm{pol}]$.

The proof of the theorem uses succinct upgrade techniques for function problems whose inputs are given in succinct circuit description. These techniques are described in detail in the following section.

6 Problems with Succinct Inputs

6.1 Previous work and methodology

A problem is succinct, if its input is not given by a string as usual, but by a Boolean circuit which computes the bits of this string. For example, a graph can be represented by a circuit with $2n$ input gates, such that on input of two binary numbers v, w of length n, the circuit outputs if there is an edge from vertex v to vertex w. In this way, a circuit of size $O(n)$ can represent a graph with 2^n vertices. Suppose that a graph algorithm runs in time polynomial in the number of vertices. Then the natural algorithm on the succinctly represented graph runs in exponential time. Similarly, upper bounds for other time and space measures can be obtained.

The question of lower bounds for succinct problems has been studied in a series of papers about circuits [35, 22, 31, 24, 3, 7, 44], and also about other forms of succinctness such as representation by Boolean formulas or OBDDs [42, 43]. The first crucial step in these results is a so-called *conversion lemma*. It states that reductions between ordinary problems can be lifted to reductions between succinct problems:

Conversion Lemma *If $A \leq^X B$ then $s(A) \leq^Y s(B)$.*

Here, $s(A)$ denotes the succinct version of A, X and Y denote suitable notions of reducibility, where $\leq^Y$ is transitive.

For the second step, an operator 'long' is introduced which is antagonistic to s in the sense that it reduces the complexity of its arguments. For a binary language A, $\mathrm{long}(A)$ can be taken as the set of strings w whose size $|w|$ written as a binary string is in A. Contrary to s, long contains instances which are exponentially larger than the input to A. For a complexity class C, $\mathrm{long}(C)$ is the set of languages $\mathrm{long}(A)$ for all $A \in C$. It remains to show a second lemma:

Compensation Lemma $A \leq^Y s(\mathrm{long}(A))$.

Then the following theorem can be derived:

Theorem *Let C_1, C_2 be complexity classes such that $\mathrm{long}(C_1) \subseteq C_2$, and let A be C_2-hard under $\leq^X$ reductions. Suppose that the Conversion Lemma and the Compensation Lemma holds. Then $s(A)$ is C_1-hard under $\leq^Y$ reductions.*

Proof. To show C_1-hardness, let B be an arbitrary problem in C_1. By assumption, $\mathrm{long}(B) \in C_2$, and therefore, $\mathrm{long}(B) \leq^X A$. By the Compensation Lemma, $B \leq^Y s(\mathrm{long}(B))$, and by the Conversion Lemma, we obtain $s(\mathrm{long}(B)) \leq^Y s(A)$. Since $\leq^Y$ is transitive, $s(A)$ is C_1-hard. ◄

6.2 Queries on succinct inputs

For any τ-structure $\mathcal{A}$, let $enc(\mathcal{A})$ denote the encoding of $\mathcal{A}$ by a binary string. The standard way to encode $\mathcal{A}$ is to fix an order on the domain elements, and to concatenate the characteristic sequences of all relations in $\mathcal{A}$.[2] All Turing machine based algorithms (and in particular, all reductions) in fact work on $\mathcal{A}$. Therefore, we shall usually identify $\mathcal{A}$ and $enc(\mathcal{A})$ without further notice. We use the further notation:

- $enc(\tau)$ denotes the binary language of all encodings of finite τ-structures.
- $\mathrm{char}(\mathcal{A})$ is the value of the binary number obtained by concatenating a leading 1 with $enc(\mathcal{A})$.

Given a binary circuit $\mathbf{C}$ with k input gates, $gen(\mathbf{C})$ denotes the binary string of size 2^k obtained by evaluating the circuit for all possible assignments in lexicographical order.

The idea of succinct representation is to represent $enc(\mathcal{A})$ in the form $gen(\mathbf{C})$. To overcome the mismatch between the fact that the size of $enc(\mathcal{A})$ can be almost arbitrary, while the size of $gen(\mathbf{C})$ has always the form 2^k, we use self-delimiting encodings:

Definition 6.1 *Let $w = (x_1, \ldots, x_n) \in \{0,1\}^+$. The* self-delimiting *encoding of w is defined as $sd(w) = (x_1, 0, x_2, 0, \ldots, x_{n-1}, 0, x_n, 1)$. For a number n, $sd(n) = sd(\overline{n})$, where $\overline{n}$ denotes the binary representation of the number n.*

Thus, from a string $sd(w)v$, the string w can be easily retrieved by looking for the first 1 at an even position in the string.

Definition 6.2 ([42]) *For a binary language L, let $sd(L)$ denote the language*

$$\{sd(|w|)wv \ : \ w \in L, |sd(|w|)wv| = 2^r, r > 1\}.$$

[2] The characteristic sequence of a relation is the binary string which for all tuples in lexicographical enumeration describes membership in the relation by 1, and non-membership by 0; for graphs, this means writing down the adjacency matrix line by line.

Thus, $sd(L)$ is the language obtained from L by adding the length descriptor and then some dummy string that pads its size to a power of 2.

Definition 6.3 *An FPLT function f is computed by two polylogarithmic time bounded deterministic Turing Machines N and M, such that on input x, N computes the size of the output $|f(x)|$, and on input x and i, M computes the i-th bit of $f(x)$.*

A PLT reduction is a reduction computed by an FPLT function.

Modulo PLT reductions, self-delimiting encoding is equivalent to standard encoding:

Lemma 6.1 ([42]) *For a nonempty binary language L, $L \equiv^{\mathrm{PLT}} sd(L)$.*

In particular, this means that there exists an FPLT function extract, which extracts a word from its self-delimiting encoding.

Definition 6.4 *Let F be a query on τ-structures. The succinct version $s(F)$ of F is given by*

$$s(F) = \{(\mathbf{C}, F(\mathcal{T})) : \mathrm{extract}(gen(\mathbf{C})) = \mathcal{T}\}$$

If $gen(\mathbf{C}) \in sd(enc(\tau))$, let $gen_\tau(\mathbf{C})$ denote the corresponding τ-structure, otherwise $gen_\tau(\mathbf{C})$ denotes some default τ-structure.

Using gen_τ, we can rephrase the definition of $s(F)$ as follows:

$$s(F) = \{(\mathbf{C}, F(gen_\tau(\mathbf{C})) : \mathbf{C} \text{ is a circuit}\}$$

The weak reducibility needed for the antecedent of the conversion lemma is given by so-called forgetful metric reductions; they differ from metric reductions in that the complexity of the inner function is restricted to FPLT, and that the outer ("forgetful") function may not access the original input.

Definition 6.5 *A function f is* **forgetfully metric reducible** *to a function g (in symbols, $f \leq_f^{mr} g$), if there is an FPLT function h_1 and a polynomial time computable function h_2 such that for every x, $f(x) = h_2(g(h_1(x)))$.*

It is not hard to see that $\leq_f^{mr}$ is transitive. The crucial observation needed to generalize the results about succinct decision problems to succinct function problems is that the succinct representation affects only the inner computation in the metric reduction (i.e., h_1), because the result of the succinct function $s(F)$ is not succinct. Thus, if we are able to lift the inner reductions from ordinary instances to succinct instances, then we can leave the outer computation (i.e., h_2) unchanged. This lifting is achieved by the following lemma:

Lemma 6.2 (immediate from [44]) *Let f be a FPLT function which maps τ-structures to σ-structures. Then there exists an FPLT function F s.t. for all circuits $\mathbf{C}$*

$$f(gen_\tau(\mathbf{C})) = gen_\sigma(F(\mathbf{C}))$$

With this background, the conversion lemma is easy to show:

Lemma 6.3 (Conversion Lemma) *Let F be a query over τ-structures, and G be a query over σ-structures. If $F \leq_f^{mr} G$ then $s(F) \leq_f^{mr} s(G)$.*

Proof. By assumption we have $F(\mathcal{A}) = h_2(G(h_1(\mathcal{A})))$. We have to show that there exist an FPLT function H_1 and a polynomial time function H_2 such that

$$[(s(F)](\mathbf{C}) = H_2(\mathbf{C}, [s(G)](H_1(\mathbf{C}))).$$

By Lemma 6.2 there is an H_1 s.t. $h_1(gen_\tau(\mathbf{C})) = gen_\sigma(H_1(\mathbf{C}))$, and thus,

$$[s(G)](H_1(\mathbf{C})) = G(h_1(gen_\tau(\mathbf{C}))).$$

Then we can set $H_2 = h_2$, and the lemma is proven. ◀

It remains to define a suitable long operator. Recall that it has to simplify the complexity of its argument. Following [42], we obtain the following definition for long on queries:

Definition 6.6 *Let $\delta = (R^1)$ be a signature with a single unary relation symbol, and let Q be a convex query over signature τ. Then the query $\mathrm{long}(Q)$ is defined as follows:*

$$\mathrm{long}(Q) = \{(\mathcal{D}, Q(\mathcal{T})) : \mathcal{D} \textit{ is a } \delta\textit{-structure}, \mathrm{char}(\mathcal{T}) = |R^{\mathcal{D}}|\},$$

where $\mathrm{char}(\mathcal{A})$ is the value of the binary number obtained by concatenating a leading 1 with the characteristic sequence of the tuples in $\mathcal{A}$ in lexicographical order.

Lemma 6.4 (Compensation Lemma) *Let F be a query. Then $F \leq_f^{mr} s(\mathrm{long}(F))$.*

Proof. As in Lemma 6.3, it is sufficient to show that every input $\mathcal{T}$ of F can be translated into a circuit $\mathbf{C}$ s.t. $|R^{gen_\delta(\mathbf{C})}| = \mathrm{char}(\mathcal{T})$. This was shown (using somewhat different terminology) in [42, Lemma 6]. ◀

Theorem 6.5 *Let $\mathbf{F}_1, \mathbf{F}_2$ be two classes of functions, such that $\mathrm{long}(\mathbf{F}_1) \subseteq \mathbf{F}_2$. If a query F is hard for $\mathbf{F}_2$ under $\leq_f^{mr}$-reductions, then $s(F)$ is hard for $\mathbf{F}_1$ under $\leq_f^{mr}$-reductions.*

6.3 Succinctness and expression complexity

Succinct problems and expression complexity are related by the following methods, which was used in [23, 14] and generalized in [18]:

Suppose that a language L can express a C_2-complete property A. Then its data complexity is trivially C_2. If the language is rich enough to simulate a Boolean circuit by a program of roughly the same size, then it is possible to combine a program for A with a program for circuit simulation, thus obtaining a program for $s(A)$. Consequently, there is a reduction from $s(A)$ to the expression complexity of L.

In [14], it was shown how a negation-free DDDB can simulate a Boolean circuit: Let $C = \{g_i = (a_i, j, l), 1 \leq i \leq t\}$ be a boolean circuit that decides a k-ary predicate R over $\{0, 1\}$, i.e., for any tuple

$\bar{t} \in \{0,1\}^k$ supplied to C as input, a designated output gate of C, which we assume is g_t, has value 1 iff $\bar{t} \in R$.

We describe a program π_C that simulates C using the universe $\{0,1\}$. For each gate g_i, π_C uses a k-ary predicate G_i, where $G_i(\bar{x})$ informally states that on input of tuple $\bar{x}$ to C, the circuit computation sets the output of g_i to 1. Moreover, it uses a propositional letter *False*, which is true in those models in which the G_i do not have the intended interpretation; none of these models will be minimal.

The clauses of π_C are the following ones. For each gate $g_i = (a_i, j, l)$ of C, it contains the clause

$$(00) \quad G_i(\bar{x}) \leftarrow \mathit{False}.$$

Depending on the type a_i, π_C contains for g_i additional clauses:

$$\begin{array}{lll}
a_i = in: & (01) & G_i(x_1, \ldots, x_{j-1}, 1, x_{j+1}, \ldots, x_k) \leftarrow\ . \\
& (02) & \qquad \mathit{False} \leftarrow G_i(x_1, \ldots, x_{j-1}, 0, x_{j+1}, \ldots, x_k). \\
& & \\
a_i = not: & (03) & G_i(\bar{x}) \vee G_j(\bar{x}) \leftarrow\ . \\
& (04) & \qquad \mathit{False} \leftarrow G_j(\bar{x}), G_i(\bar{x}). \\
& & \\
a_i = and: & (05) & \qquad G_i(\bar{x}) \leftarrow G_j(\bar{x}), G_l(\bar{x}). \\
& (06) & \qquad G_j(\bar{x}) \leftarrow G_i(\bar{x}). \\
& (07) & \qquad G_l(\bar{x}) \leftarrow G_i(\bar{x}). \\
& & \\
a_i = or: & (08) & \qquad G_i(\bar{x}) \leftarrow G_j(\bar{x}). \\
& (09) & \qquad G_i(\bar{x}) \leftarrow G_l(\bar{x}). \\
& (10) & G_j(\bar{x}) \vee G_l(\bar{x}) \leftarrow G_i(\bar{x}).
\end{array}$$

The clauses (00) ensure that if a model of $ground(\pi_C, \{0,1\})$ contains *False*, it is the maximal interpretation (which is trivially a model of π_C). In fact, this is the only model of π_C that contains *False*. Let M_C denote the interpretation given by $M_C = \{G_i(\bar{t}) \mid g_i \in C, \bar{t} \in \{0,1\}^k$, and g_i takes value 1 on input $\bar{t}$ to C $\}$.

Lemma 6.6 ([14]) *For any Boolean circuit C, M_C is the unique minimal model of π_C.*

Theorem 6.7 *Problem λDD is complete for* $\mathrm{FPSpace}^{\mathrm{NP}}$.

Proof. Define as usual a problem whose input contains a uniform circuit with constant input gates which generates the instance to be solved. Then it is easy to see that $\mathrm{FP}_{\|}^{\mathrm{NP}}$ contains a query Q which is complete under $\leq_f^{mr}$-reductions. (For example, QUERY is such a problem: the function h_1 of any metric reduction $g \leq^{mr}$ QUERY can be shifted inside the oracle queries in polylog-time, and the bits of the input string x provided through dummy oracle queries to h_2.) It is not hard to see that $\mathrm{long}(\mathrm{FLinSpace}^{\mathrm{NP}}) \subseteq \mathrm{FP}_{\|}^{\mathrm{NP}}$, and therefore by Theorem 6.5, $s(Q)$ is complete for $\mathrm{FLinSpace}^{\mathrm{NP}}$. By standard padding arguments, completeness for $\mathrm{FLinSpace}^{\mathrm{NP}}$ implies completeness for $\mathrm{FPSpace}^{\mathrm{NP}}$. Thus, it remains to reduce $s(Q)$ to λDD.

By Lemma 6.6, a circuit $\mathbf{C}$ with k input gates can be converted into a disjunctive program $\pi_{\mathbf{C}}$ whose k-ary output relation R describes the string $gen(\mathbf{C})$. Consider the query $Q(\mathrm{extract}(\mathcal{A}))$, where $\mathcal{A}$ is an ordered input structure which describes a string by a unary relation. Since $Q(\mathrm{extract}(\mathcal{A}))$ is easily seen to be in $\mathrm{FP}_{\|}^{\mathrm{NP}}$, Theorem 5.1 implies that there is a program π whose data disjunction describes the result of $Q(\mathrm{extract}(\mathcal{A}))$. Since DDDBs are uniformly closed, π can be rewritten into a program π' whose input relation R has arity k. As in the proof of Theorem 5.1 we can assume that there is a lexicographical successor relation on k-tuples. By well-known modularity properties [14, Section 5], the program $\pi' \cup \pi_{\mathbf{C}}$ indeed computes Q on the succinctly specified input. ◀

It is not hard to show that also $\mathrm{FL}_{\log}^{\mathrm{NP}}$ has $\leq_f^{mr}$-complete queries (e.g., a variant of CLIQUE SIZE in which circuits computing the functions h_1, h_2 of a metric reduction to CLIQUE SIZE are part of the problem instance). The following theorem is then shown analogously:

Theorem 6.8 *Problem k-λDD is complete for* $\mathrm{FPSpace}^{\mathrm{NP}}[\mathrm{pol}]$.

7 Further Results and Conclusion

7.1 Closed world reasoning

The results on data disjunctions that we have derived above have an immediate application to related problems in the area of closed-world reasoning.

Reiter [36] has introduced the closed-world assumption (CWA) as a principle for inferring negative information from a logical database. Formally,

$$CWA(\mathsf{DB}) = \mathrm{ground}(\mathsf{DB}) \cup \{\neg A \mid A \in \mathsf{HB}_{\mathsf{DB}} \text{ and } \mathsf{DB} \not\models A\}.$$

For example, $CWA(\{P(a), Q(b)\}) = \{P(a), Q(b), \neg P(b), \neg P(a)\}$. It follows from results in [11] that computing $CWA(\mathsf{DB})$ has propositional complexity $\mathrm{FP}_{\|}^{\mathrm{NP}}$.

Observe that $CWA(\mathsf{DB})$ may not be classically consistent (under Herbrand interpretations); for example, $CWA(\{P \vee Q\}) = \{P \vee Q, \neg P, \neg Q\}$ which has no model. As shown in [11], deciding whether $CWA(\mathsf{DB})$ is consistent is in Θ_2^{P} and coNP-hard in the propositional case; the precise complexity of this problem is open.

In a refined notion of partial CWA (cf. [16]), which is in the spirit of protected circumscription [34], only atoms A involving a particular predicate P or, more general, a predicate P from a list of predicates $\mathbf{P}$ may be negatively concluded from DB:

$$PCWA(\mathsf{DB}; \mathbf{P}) = \mathrm{ground}(\mathsf{DB}) \cup \{\neg P(\bar{c}) \mid P(\bar{c}) \in \mathsf{HB}_{\mathsf{DB}} \text{ and } P \in \mathbf{P} \text{ and } \mathsf{DB} \not\models A\}.$$

E.g., $CWA(\{P(a), Q(b)\}; P) = \{P(a), Q(b), \neg P(b)\}$.

Definition 7.1 (P-minimal model) *Let $\mathbf{P}$ be a list of predicates. The preorder $\preceq_{\mathbf{P}}$ on the models of a DDDB* DB *is defined as follows: $M \preceq_{\mathbf{P}} M' \Leftrightarrow$ for every $P(\bar{c}) \in \mathsf{HB}_{\mathsf{DB}}$ such that $P \in \mathbf{P}$ it holds that $P(\bar{c}) \in M \Rightarrow P(\bar{c}) \in M'$. A model M is $\mathbf{P}$-minimal for* DB*, if there exists no model M' such that $M' \prec M$, where $M' \prec M$ means $M' \preceq_{\mathbf{P}} M \wedge M \not\preceq_{\mathbf{P}} M'$.*

We remark that a $\mathbf{P}$-minimal model is a special case of the notion of $\mathbf{P}; \mathbf{Z}$ minimal model [28], given by an empty list of fixed predicates in a circumscription.

Proposition 7.1 *Let* DB *be a DDDB and* P *a predicate. Then, the following statements are equivalent:*

1. DB *has a data disjunction on* P.
2. $PCWA(\mathsf{DB}; P)$ *is not consistent (with respect to classical Herbrand models).*
3. DB *does not have a global* P*-minimal model* M*, i.e.,* $M \preceq_P M'$ *for all models* M' *of* DB.

Proof. 1. $\rightarrow$ 2.: Suppose $C = P(\bar{c}_1) \vee \cdots \vee P(\bar{c}_n)$, $n \geq 2$, is a data disjunction of DB. Then, $\mathsf{DB} \not\models P(\bar{c}_i)$, which means $\neg P(\bar{c}_i) \in PCWA(\mathsf{DB}; P)$, for every $i = 1, \ldots, n$. $\mathsf{DB} \models C$ implies that $PCWA(\mathsf{DB}; P) \models C$, which means $PCWA(\mathsf{DB}; P)$ is not consistent.

$\neg 3. \rightarrow \neg 2.$: Suppose DB has a global P-minimal model M. Then, for each atom $P(\bar{c}) \in \mathsf{HB}_{\mathsf{DB}}$ it holds that $\mathsf{DB} \not\models P(\bar{c})$ iff $M \not\models P(\bar{c})$, since M has the unique smallest P-part over all models of DB. Hence, M is model of $PCWA(\mathsf{DB}; P)$, and thus $PCWA(\mathsf{DB}; P)$ is consistent.

3. $\rightarrow$ 1.: Suppose DB has no global P-minimal model. Let $M_1, \ldots, M_n$ be the collection of all P-minimal models M of DB, where w.l.o.g. $M_1 \not\preceq_P M_2$ and $M_2 \not\preceq_P M_1$. Let X be the set of all atoms in $M_1 \cup M_2$ of form $P(\bar{c})$, and let $P(\bar{c}_1) \in M_1 \setminus M_2$ and $P(\bar{c}_2) \in M_2 \setminus M_1$ be arbitrary atoms. Then, $M_i \models C$ where $C = \bigvee\{P(\bar{c}) \mid P(\bar{c}) \in (\mathsf{HB}_{\mathsf{DB}} \setminus X) \cup \{P(\bar{c}_1), P(\bar{c}_2)\}\}$ holds for every $i = 1, \ldots, n$, which means $\mathsf{DB} \models C$. Obviously, $\mathsf{DB} \not\models C \setminus P(\bar{c}_1)$ and $\mathsf{DB} \not\models P(\bar{c}_2)$. Thus, C contains a data disjunction on predicate P (which contains $P(\bar{c}_1) \vee P(\bar{c}_2)$). ◀

As for $\mathbf{P}$-minimality, a list $\mathbf{P}$ of predicates can, by simple coding, be replaced with a single predicate P: replace atoms $P_1(x, y)$ and $P_2(z)$ etc with $P(1, x, y)$, $P(2, z)$ etc where the first argument in P codes the predicate. This coding is compatible with $\mathbf{P}$-minimality, i.e., $\mathbf{P}$-minimal and P-minimal models correspond as obvious. From Proposition 7.1 and the results of the previous sections, we thus obtain the following result.

Theorem 7.2 *Deciding consistency of* $PCWA(\mathsf{DB}; \mathbf{P})$ *and existence of some global* $\mathbf{P}$*-minimal model of* DB *have both* Θ_2^{P} *propositional and data complexity, and* $\mathrm{PSpace}^{\mathrm{NP}}$ *program and expression complexity.*

By the same coding technique, the result in Theorem 7.2 holds even if the language has only two predicates and $\mathbf{P}$ contains a single predicate. On the other hand, if the language has a only one predicate P, then the existence of a data disjunction on p is equivalent to the consistency of $CWA(\mathsf{DB})$, whose precise complexity is open.

7.2 Restricted data disjunctions

In [15], a stronger notion of data disjunction $R\bar{c}_1 \vee \cdots \vee R\bar{c}_k$ is considered, which requests in addition that all disjuncts $R\bar{c}_i$ are identical up to one argument of the list of constants $\bar{c}_i$; we call such data disjunctions *restricted.* Note that all data disjunction considered in Section 1 are restricted.

For the problems reformulated to restricted data disjunctions, Table 1 in Section 1 is the same except that the expression and combined complexity of λDD is $\mathrm{FPSpace}^{\mathrm{NP}}[\mathrm{pol}]$. Indeed, a restricted data disjunction C has at most m disjuncts where $m = |\mathsf{HU}_{\mathsf{DB}}|$ is the number of constants, and thus λDD has $O(n^2 \log n)$ many output bits in the combined complexity case, where n is the size of DB. The number of maximal disjunctions $\mathrm{md}(\mathsf{DB}, R)$, adapted to restricted data disjunctions, is polynomial in the data size, and thus the same upper bounds can be easily derived as for unrestricted data disjunctions. All hardness results are immediate from the proofs except for propositional and data complexity of λDD; here, mapping tuples $\bar{c}$ of elements to newly introduced (polynomially many) domain elements is a suitable technique for adapting the construction in Figure 2 in the proof of Theorem 5.1.

Finally, we remark that Lemma 2.3 remains true even if all disjunctions in the program π_φ describe restricted data disjunctions. Thus, by a slight adaptation of the programs in proofs and exploiting the fact that disjunction-free datalog with input negation is sufficient for upgrading purposes [14], the complexity results for restricted data disjunction remain true even if all disjunctions in DB must be restricted data disjunctions.

7.3 Conclusion

In this paper, we have considered the complexity of some problems concerning data disjunctions in deductive databases. To this aim, we have taken an "engineering perspective" on deriving complexity results using tools from the domain of descriptive complexity theory, and combined them with results for upgrading complexity results on normal to succinct representations of the problem input. In particular, we have also investigated the complexity of actually computing data disjunctions as a function, rather than only the associated decision problem. This led us to generalize upgrading techniques developed for decision problems to computations of functions. These upgrading results, in particular Theorem 6.5, may be conveniently used in other contexts.

The tools as used and provided in this paper allow for a high-level analysis of the complexity of problems, in the sense that establishing certain properties and schematic reductions are sufficient in order to derive intricate complexity results as eg. for the case of data disjunctions in a clean and transparent way, without the need to deal with particular problems in reductions. While this relieves us from spelling out detailed technical constructions, the understanding of what makes the problem computationally hard may be blurred. In particular, syntactical restrictions under which the complexity remains the same or is lowered can not be immediately inferred. We leave such considerations for further work. Another interesting issue for future work is the consideration of computing data disjunctions viewed as a multi-valued function, which we have not done here.

Acknowledgment We are grateful to Iain Stewart and Georg Gottlob for discussions and remarks.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] V. Arvind and J. Torán. Sparse sets, approximable sets, and parallel queries to NP. In C. Meinel and S. Tison, editors, *Proc. Sixteenth Symposium on Theoretical Aspects of Computing (STACS '99)*, LNCS 1563, pp. 281–290, Trier, March 1999. Springer.

[3] J. Balcázar, A. Lozano, and J. Torán. The complexity of algorithmic problems on succinct instances. In R. Baeta-Yates and U. Manber, editors, *Computer Science*, pp. 351–377. Plenum Press, New York, 1992.

[4] R. Beigel. NP-hard sets are P-superterse unless R=NP. Technical Report TR4, CS Dept., Johns Hopkins University, 1988.

[5] P. Bonatti. Autoepistemic logics as a unifying framework for the semantics of logic programs. *J. Logic Programming*, 22:91–149, 1995.

[6] P. A. Bonatti and T. Eiter. Querying disjunctive databases through nonmonotonic logics. *Theoretical Computer Science*, 160:321–363, 1996.

[7] B. Borchert and A. Lozano. Succinct circuit representations and leaf languages are basically the same concept. *Information Processing Letters*, 58:211–215, 1996.

[8] H. Buhrmann, L. Fortnow, and L. Torenvliet. Six hypotheses in search of a theorem. In *Proc. 12th IEEE International Conference on Computational Complexity (CCC '97)*, pp. 2–12, 1997.

[9] M. Cadoli and M. Lenzerini. The complexity of propositional closed world reasoning and circumscription. *J. Computer and System Sciences*, 43:165–211, April 1994.

[10] J. Dix. Semantics of logic programs: Their intuitions and formal properties. an overview. In A. Fuhrmann and H. Rott, editors, *Logic, Action and Information. Proc. Konstanz Colloquium in Logic and Information (LogIn'92)*, pp. 241–329. DeGruyter, 1995.

[11] T. Eiter and G. Gottlob. Propositional circumscription and extended closed world reasoning are Π_2^P-complete. *Theoretical Computer Science*, 114(2):231–245, 1993. Addendum 118:315.

[12] T. Eiter and G. Gottlob. The complexity class $\Theta_2^{\mathbf{p}}$: Recent results and applications in AI and modal logic. In B. Chlebus and L. Czaja, editors, *Proc. 11th International Symposium on Fundamentals of Computation Theory (FCT '97)*, LNCS 1279, pp. 1–18. Springer, 1997.

[13] T. Eiter, G. Gottlob, and Y. Gurevich. Normal forms for second-order logic over finite structures, and classification of NP optimization problems. *Annals of Pure and Applied Logic*, 78:111–125, 1996.

[14] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Transactions on Database Systems*, 22(3):364–417, September 1997.

[15] Z. K. J. Fernández and J. Minker. A tractable class of disjunctive deductive databases. In *Proc. Workshop on Deductive Databases, JICSLP-92*, pp. 11–20, Washington DC, 1992. Technical Report CITRI/TR-92-65, CS Dept, Univ. Melbourne.

[16] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[17] G. Gottlob. Relativized logspace and generalized quantifiers over ordered finite structures. *J. Symbolic Logic*, 62(2):545–574, 1997.

[18] G. Gottlob, N. Leone, and H. Veith. Succinctness as a source of complexity in logical formalisms. *Annals of Pure and Applied Logic*, 97:231–260, 1999.

[19] H. Hwang, A. Fu, and H.-F. Leung. Data-disjunctive deductive databases. Manuscript, 1997.

[20] T. Imielinski and K. Vadaparty. Complexity of query processing in databases with or-objects. In *Proc. 8th ACM Symposium on Principles of Database Systems (PODS '89)*, pp. 51–65, 1989.

[21] B. Jenner and J. Toran. Computing functions with parallel queries to NP. *Theoretical Computer Science*, 141:175–193, 1995.

[22] M. Karpinski and K. Wagner. The computational complexity of graph problems with succinct multigraph representation. *Zeitschrift für Operations Research*, 32:201–211, 1988.

[23] P. Kolaitis and C. H. Papadimitriou. Why not negation by fixpoint ? *J. Computer and System Sciences*, 43:125–144, 1991.

[24] M. Kowaluk and K. Wagner. Vector language: Simple descriptions of hard instances. In *Mathematical Foundations of Computer Science (MFCS)*, volume 452 of *LNCS*, pp. 378–384. Springer, 1992.

[25] M. Krentel. The complexity of optimization problems. *J. Computer and System Sciences*, 36:490–509, 1988.

[26] R. Ladner and N. Lynch. Relativization questions about logspace computability. *Mathematical Systems Theory*, 10:19–32, 1976.

[27] R. Ladner, N. Lynch, and A. Selman. Comparison of polynomial-time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.

[28] V. Lifschitz. Computing circumscription. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI-85)*, pp. 121–127, 1985.

[29] J. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1984, 1987.

[30] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, Cambridge, MA, 1992.

[31] A. Lozano and J. Balcázar. The complexity of graph problems for succinctly represented graphs. In *Proc.* 15^{th} *Intl. Workshop on Graph-Theoretic Concepts in Computer Science*, LNCS 411, pp. 277–286, Castle Rolduc, The Netherlands 1989.

[32] J. Minker. On indefinite data bases and the closed world assumption. In D. Loveland, editor, *Proc.* 6^{th} *Conference on Automated Deduction (CADE '82)*, LNCS 138, pp. 292–308, New York, 1982. Springer.

[33] J. Minker. Logic and databases: A 20 year retrospective. In *Proc. International Workshop on Logic in Databases (LID '96)*, LNCS 1154, pp. 3–57. Springer, 1996.

[34] J. Minker and D. Perlis. Computing protected circumscription. *J. Logic Programming*, 2(4):235–249, 1985.

[35] C. Papadimitriou and M. Yannakakis. A Note on succinct representations of graphs. *Information and Computation*, 71:181–185, 1985.

[36] R. Reiter. On closed-world databases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pp. 55–76. Plenum Press, New York, 1978.

[37] A. Selman. A taxonomy of complexity classes of functions. *J. Computer and System Sciences*, 48:357–381, 1994.

[38] I. Stewart. Logical characterizations of bounded query classes I: Logspace oracle machines. *Fundamenta Informaticae*, 18:65–92, 1993.

[39] I. Stewart. Logical characterizations of bounded query classes II: Polynomial-time oracle machines. *Fundamenta Informaticae*, 18:93–105, 1993.

[40] S. Toda. On polynomial-time truth-reducibilities of intractable sets to p-selective sets. *Mathematical Systems Theory*, 24:69–82, 1991.

[41] M. Vardi. The complexity of relational query languages. In *Proc. 14th ACM Symposium on the Theory of Computing (STOC '82)*, pp. 137–146, San Francisco, 1982.

[42] H. Veith. Languages represented by Boolean formulas. *Information Processing Letters*, 63:251–256, 1997.

[43] H. Veith. How to encode a logical structure by an obdd. In *Proc. 13th Annual IEEE Conference on Computational Complexity (CCC '98)*, pp. 122–131, 1998.

[44] H. Veith. Succinct representation, leaf languages, and projection reductions. *Information and Computation*, 142(2):207–236, 1998.

[45] K. Wagner. Bounded query classes. *SIAM Journal of Computing*, 19(5):833–846, 1990.