

A Negative Conjunctive Query is Easy if and only if it is Beta-Acyclic

Johann Brault-Baron

Université de Caen/ENSICAen/CNRS - GREYC (UMR6072)
Bd Maréchal Juin 14050 Caen Cedex, France
Johann.Brault-Baron@unicaen.fr

Abstract

It is known that the data complexity of a Conjunctive Query (CQ) is determined *only* by the way its variables are shared between atoms, reflected by its hypergraph. In particular, Yannakakis [18, 3] proved that a CQ is decidable in linear time when it is α -acyclic, i.e. its hypergraph is α -acyclic; Bagan et al. [2] even state:

- *Any CQ is decidable in linear time iff it is α -acyclic. (under certain hypotheses)*
- (By linear time, we mean a query on a structure \mathcal{S} can be decided in time $\mathcal{O}(|\mathcal{S}|)$)

A natural question is: since the complexity of a *Negative* Conjunctive Query (NCQ), a conjunctive query where *all* atoms are negated, also only depends on its hypergraph, can we find a similar dichotomy in this case?

To answer this question, we revisit a result of Ordyniak et al. [17] — that states that satisfiability of a β -acyclic CNF formula is decidable in polynomial time — by proving that some part of their procedure can be done in linear time. This implies, under an algorithmic hypothesis (precisely: one cannot decide whether a graph is triangle-free in time $\mathcal{O}(n^2 \log n)$ where n is the number of vertices.) that is likely true:

- *Any NCQ is decidable in quasi-linear time iff it is β -acyclic.*

(By quasi-linear time, we mean a query on a structure \mathcal{S} can be decided in time $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$)

We extend the easiness result to *Signed* Conjunctive Query (SCQ) where *some* atoms are negated. This has great interest since using some negated atoms is natural in the frameworks of databases and CSP. Furthermore, it implies straightforwardly the following:

- *Any β -acyclic existential first-order query is decidable in quasi-linear time.*

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, H.2.4 Query processing

Keywords and phrases conjunctive query, hypergraph, β -acyclicity, data complexity, Davis-Putnam procedure.

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.137

1 Introduction

This paper gives descriptive complexity results in a finite model theory framework. According to [16], “Finite model theory studies the expressive power of logic on finite relational structures.” Here we emphasize the complexity aspect of expressive power, in relation with the considered (first-order) logic fragment.

A fragment of great interest is the primitive positive fragment, i.e. the set of sentences one can build using only atoms — relations between variables —, the conjunction \wedge and the existential quantifier \exists . This fragment, also known as Conjunctive Queries (CQ), is fundamental in database theory (see [5] for example) and in Constraint Satisfaction Problems (CSP) (see [7]). While general queries (first-order sentences) are PSPACE-complete in terms



© Johann Brault-Baron;
licensed under Creative Commons License NC-ND
Computer Science Logic 2012 (CSL'12).

Editors: Patrick Cégielski, Arnaud Durand; pp. 137–151



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of combined complexity, conjunctive queries are “only” NP-complete in terms of combined complexity, or W[1]-Complete ([13]) in terms of parametrized complexity.

An important tractable class of conjunctive queries are the α -acyclic (or *acyclic*, for short) conjunctive queries. They are tractable in a very strong sense: Yannakakis [18] proved that an α -acyclic conjunctive query ϕ on a relational structure \mathcal{S} can be decided in time $\mathcal{O}(|\phi| \cdot |\mathcal{S}|)$; in particular, it is fixed-parameter linear, i.e. it has linear data complexity. Bagan et al. ([2]) even states (partially) that α -acyclicity is a necessary condition of linear data complexity. Another result giving relevance to this class is that Gottlob et al. ([10]) proved α -acyclic CQ to be LOGCFL-complete for combined complexity.

Acyclic CQ have met a great interest essentially because they are the basis of important tractable classes of queries: they have been extended to several notions of queries not-too-far from α -acyclicity. In particular, this notion was extended to bounded treewidth CQ — a notion of bounded distance from being a tree — leading to classification results, see [13]; and to the notion of bounded hyper-treewidth CQ[11], for example. These classes allow polynomial reduction to acyclic CQ.

Nevertheless, in a database context, polynomiality is not a sufficient notion of tractability: a quadratic dependency on the database size is often considered intractable. A more reasonable notion would be quasi-linear time $n(\log n)^{\mathcal{O}(1)}$ (see [14]), which we take as a definition of tractability. Notice that, according to this definition of tractability, tractable CQ are still exactly CQ that have an α -acyclic hypergraph.

One can naturally ask oneself which queries, besides α -acyclic CQ, are tractable. Notice that there are some obviously tractable queries that are not in CQ, in particular some sets operations; e.g., the query $\exists x R(x) \wedge \neg S(x)$ that can be interpreted as “is $R \setminus S$ non-empty?”. This example leads to a quite natural question: which extensions of CQ where *some* atoms are negated, which we call *Signed Conjunctive Queries* (SCQ), are tractable?

In order to answer this question, we investigate a simpler question: Which queries that are conjunctions of only negated atoms, that we call *Negative Conjunctive Queries* (NCQ), are tractable? The advantage of this simpler case is that we can use the same tool as the known CQ case: the notion of hypergraph. The hypergraph of a query is quite a simple object reflecting the way variables are shared between atoms. It is a widespread intuitive idea that the complexity of a CQ only depends on its hypergraph, and in fact it is easy to prove this also holds in the case of NCQ.

One might think that a NCQ can be immediately reduced to a CQ by complementing the relations. However, computing the complement is clearly not in FPT, and *a fortiori* not in quasi-linear time.

We will prove for this case a dichotomy similar to the one known for CQ: a NCQ is tractable iff it is β -acyclic, which means that its hypergraph is β -acyclic. We finally extend the easiness result by proving that β -acyclic SCQ are tractable.

Structure of this Paper

This paper is organized as follows:

- Section 1 introduces the main definitions and states the results;
- Section 2 refines a result from Ordyniak et al. [17] by proving Davis-Putnam resolution w.r.t. a variable that is a nest point is done in linear time;
- Section 3 proves the easiness result;
- Section 4, after introducing some technical points, establishes the hardness result.

2 Preliminaries and Results

We introduce here the respective statements of the two results with all necessary definitions.

2.1 Preliminaries

► **Definition 1** (sentence, structure, query, CQ, NCQ, SCQ). A *signature* σ is:

- a set of relation symbols,
- and an arity Ar that associates a number to each symbol R , denoted $\text{Ar}(R)$.

A σ -*structure* \mathcal{S} consists in associating a set of $\text{Ar}(R)$ -tuples to each of the relation symbols R of σ which is called the *interpretation of R in \mathcal{S}* , and denoted $R^{\mathcal{S}}$. Some relation symbols of arity 1 may be used in a particular way, and are then called *domain symbols*.

An *existential first-order sentence*, or *sentence* for short, is a usual existential first-order sentence where each variable has a *distinct* associated domain symbol D_i . More formally, a σ -sentence has the form $\exists x_1 \in D_1 \dots \exists x_n \in D_n \psi$ where D_i is a domain symbol belonging to σ and ψ is a usual quantifier-free σ' -formula where σ' is σ without the D_1, \dots, D_n previously mentioned. In the whole document, ψ will refer to the quantifier-free part of ϕ .

We call query of a sentence ϕ , denoted $\text{DecideQ}(\phi)$, the problem of, given a structure \mathcal{S} , deciding whether ϕ holds in \mathcal{S} . Depending on the quantifier-free formula ψ , we define classes of queries of interest:

- when $\psi = \bigwedge_i R_i(x_{i_0}, \dots, x_{i_{\text{Ar}(R_i)}})$, $\text{DecideQ}(\phi) \in \text{CQ}$ the *existential Conjunctive Queries*;
- when $\psi = \bigwedge_i \neg R_i(x_{i_0}, \dots, x_{i_{\text{Ar}(R_i)}})$, $\text{DecideQ}(\phi) \in \text{NCQ}$ the *existential Negative Conjunctive Queries*;
- when $\psi = \bigwedge_i \sigma_i R_i(x_{i_0}, \dots, x_{i_{\text{Ar}(R_i)}})$, where σ_i is either \neg or ϵ (nothing), $\text{DecideQ}(\phi) \in \text{SCQ}$ the *existential Signed Conjunctive Queries*;
- when ψ is unrestricted, i.e. written using any connectives ($\wedge, \vee, \rightarrow, \neg$, etc.) $\text{DecideQ}(\phi) \in \text{EQ}$ the *existential Queries*.

Considering the particular form of the first three classes, we may consider these formulas as conjunctions of so-called *conjuncts*, i.e. when ψ is in the form: $\psi = \bigwedge_i \sigma_i R_i(x_{i_0}, \dots, x_{i_{\text{Ar}(R_i)}})$ each $C_i = \sigma_i R_i(x_{i_0}, \dots, x_{i_{\text{Ar}(R_i)}})$ is a conjunct. When $\sigma(i)$ is ϵ (resp. \neg), we say C_i is a *positive* (resp. *negative*) conjunct.

► **Remark.** Defining sentences with distinguished domains symbols attached to variables is a bit unusual. Let us justify it briefly through a simple example. Consider:

$$\begin{aligned}\phi_1 &= \exists x_1 \exists x_2 \exists x_3 \exists x_4 R(x_1, x_2) \wedge R(x_3, x_2) \wedge R(x_1, x_4) \wedge R(x_3, x_4) \\ \phi_2 &= \exists x_1 \exists x_2 \exists x_3 \exists x_4 R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_4, x_1) \wedge R(x_3, x_4)\end{aligned}$$

The sentences ϕ_1 and ϕ_2 obviously have the same hypergraph (see below) but $\text{DecideQ}(\phi_1)$ is obviously easy (it consists in deciding whether a directed graph has at least one edge), while $\text{DecideQ}(\phi_2)$ is presumably not as easy (it consists in deciding whether a directed graph has a circuit of size 4). Using our formalism guarantees that:

- the complexity of a CQ (or a NCQ) depends only on its hypergraph, and
- the easiness results (of α -acyclic CQ and β -acyclic SCQ) still hold with the usual definition, i.e. when the variables all have the same domain.

► **Definition 2** (complexity classes). These definitions are based on [12], see it for more details. We call QLIN (resp. LIN) the set of problems decidable in time $\mathcal{O}(n \log n)$ (resp. $\mathcal{O}(n)$) on a RAM machine where n is the size of the input. In particular, sorting is in LIN by a result of [1], and [12].

► **Remark.** This definition of QLIN looks quite restrictive: $n(\log n)^{\mathcal{O}(1)}$ would be much more reasonable, as suggested by [14]. In fact, all our results hold in both cases. We chose the restrictive one in order to put the emphasis on the easiness result.

The main object of our discourse will be how the restriction of the way variables are shared in a formula allows easy decision. The way variables are shared can be described by a very simple yet powerful notion: the hypergraph.

► **Definition 3** (hypergraph, hypergraph of a query). We call *hypergraph* \mathcal{H} a set of non-empty sets, called the *edges* of \mathcal{H} . We call $\mathcal{V}(\mathcal{H})$ the union of its edges; the elements of $\mathcal{V}(\mathcal{H})$ are called the *vertices* of \mathcal{H} .

We write $\mathcal{H}(\phi)$ the hypergraph of the sentence ϕ defined as the set of variables sets appearing in atoms of ϕ : $\mathcal{H}(\phi) = \{\text{Vars}(A) \mid A \text{ is an atom of } \phi\}$ where

$$\text{Vars}\left(R_i(x_{i_1}, \dots, x_{i_{\text{Ar}(R_i)}})\right) = \{x_{i_1}, \dots, x_{i_{\text{Ar}(R_i)}}\}$$

► **Example 4.** If we consider the SCQ $\text{DecideQ}(\phi)$ where

$$\begin{cases} \phi = \exists x_1 \in D_1 \exists x_2 \in D_2 \exists x_3 \in D_3 \exists x_4 \in D_4 \psi \\ \psi = R_1(x_1, x_2) \wedge R_2(x_2, x_3) \wedge R_3(x_1, x_2, x_3) \wedge \neg R_4(x_4, x_3) \wedge \neg R_5(x_4, x_4) \end{cases}$$

then we have $\mathcal{H}(\phi) = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_1, x_2, x_3\}, \{x_3, x_4\}, \{x_4\}\}$.

In this case, $R_1(x_1, x_2)$ is a positive conjunct. By contrast, $R_5(x_4, x_4)$ is an atom but not a conjunct in this formula, while $\neg R_5(x_4, x_4)$ is a negative conjunct.

We will see that the complexity of $\text{DecideQ}(\phi)$ depends only on $\mathcal{H}(\phi)$. We now define the criterion that discerns easy queries from hard queries.

2.2 Acyclicity Notions

We will see some interesting properties of hypergraphs are several notions of acyclicity, which are different extensions of the (classical) graph acyclicity property.

► **Definition 5** (induced hypergraph, nest point). Let \mathcal{H} be a hypergraph. We define its induced hypergraph w.r.t. $S \subseteq \mathcal{V}(\mathcal{H})$, denoted $\mathcal{H}[S]$, as follows:

$$\mathcal{H}[S] = \{e \cap S \mid e \in \mathcal{H}\} \setminus \{\emptyset\}$$

We write $\mathcal{H}[\setminus S]$ as a shorthand for $\mathcal{H}[\mathcal{V}(\mathcal{H}) \setminus S]$. In particular, $\mathcal{H}[\setminus \{x\}]$ is the hypergraph obtained by weak vertex removal of x .

We say x is a *nest point* of \mathcal{H} when for any two distinct edges e_1 and e_2 containing x , either $e_1 \subset e_2$ or $e_2 \subset e_1$. In other words, the set $\{e \in \mathcal{H} \mid x \in e\}$ is linearly ordered w.r.t. set inclusion.

Fagin's originally defined ([9]) α -acyclicity of a hypergraph \mathcal{H} , here denoted $\mathcal{A}_\alpha(\mathcal{H})$. We assume the reader is familiar with the notion of α -acyclicity, that is the most classical notion of acyclicity for hypergraphs. He also defined β -acyclicity as follows:

► **Definition 6.** We say that a hypergraph \mathcal{H} is β -acyclic, denoted $\mathcal{A}_\beta(\mathcal{H})$, if each of its subsets, i.e. every $\mathcal{H}' \subseteq \mathcal{H}$, is α -acyclic.

This characterisation says that β -acyclicity is the ‘‘hereditary’’ closure of α -acyclicity.

We give a second characterisation of β -acyclicity. This inductive characterisation from [8], based on a result of [4], is useful for the algorithm we give for β -acyclic queries (easiness result).

► **Lemma 7** (β -acyclicity — inductive characterisation). *A hypergraph \mathcal{H} is β -acyclic iff either it is empty, or such that:*

- *we can find $x \in \mathcal{V}(\mathcal{H})$ such that x is a nest point of \mathcal{H} and*
- *$\mathcal{H}[\setminus\{x\}]$ is also β -acyclic.*

We say the ordered list (x_1, \dots, x_n) of the vertices of an hypergraph \mathcal{H} is a Reverse Elimination Order (REO) of \mathcal{H} when, for all i in $\{1, \dots, n\}$, x_i is a nest point of $\mathcal{H}[\{x_1, \dots, x_{i-1}\}]$. By this characterisation, a hypergraph is β -acyclic iff it has a REO.

Here is the third characterisation of β -acyclicity. It will be used to obtain our hardness result. This characterisation uses the notion of β -cycle defined here.

► **Definition 8.** *A chordless cycle is a graph isomorphic to $\{\{x_i, x_{i+1}\} \mid 1 \leq i \leq k\} \cup \{\{x_k, x_1\}\}$ for some k . A β -cycle of a hypergraph \mathcal{H} is a subset of some induced sub-hypergraph of \mathcal{H} that is a chordless cycle:*

$$C \text{ is a } \beta\text{-cycle of } \mathcal{H} \iff \exists S \subseteq \mathcal{V}(\mathcal{H}) \ C \subseteq \mathcal{H}[S] \text{ and } C \text{ is a chordless cycle}$$

► **Lemma 9** (β -acyclicity as absence of β -cycles). *An hypergraph \mathcal{H} is β -acyclic iff it does not have a β -cycle.*

2.3 Statement of the Results

With all these notations, we are now able to state our results:

► **Theorem 10** (dichotomy). *Under hypothesis that the presence of a triangle in a graph of n vertices cannot be decided in time $\mathcal{O}(n^2 \log n)$, we have:*

$$\forall \phi \in \text{NCQ} \quad \text{DecideQ}(\phi) \in \text{QLIN} \iff \mathcal{A}_\beta(\mathcal{H}(\phi))$$

Proof. Lemma 21 proves the implication \Leftarrow (easiness result) and Lemma 30 proves the implication \Rightarrow (hardness result). ◀

► **Remark.** This is to be compared to the positive conjunctive queries dichotomy contained in [2]: for any $\phi \in \text{CQ}$ such that $\mathcal{H}(\phi)$ is 4-conformal,¹ we have:

$$\forall \phi \in \text{CQ} \quad \text{DecideQ}(\phi) \in \text{LIN} \iff \mathcal{A}_\alpha(\mathcal{H}(\phi))$$

under hypothesis we cannot decide the presence of a triangle in a graph G in time $\mathcal{O}(|G|)$.

Notice that this result also holds if we replace LIN by QLIN; in that case the hypothesis becomes “deciding the presence of a triangle in a graph is not in QLIN.”

► **Theorem 11** (easiness).

$$\forall \phi \in \text{EQ} \quad \text{DecideQ}(\phi) \in \text{QLIN} \Leftarrow \mathcal{A}_\beta(\mathcal{H}(\phi))$$

That is to say any β -acyclic existential first-order sentence is decidable in time $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$.

Let P be a property on hypergraphs. Under the previously mentioned complexity hypothesis (about the triangle problem), we have:

$$(\forall \phi \in \text{EQ} \ P(\mathcal{H}(\phi)) \Rightarrow \text{DecideQ}(\phi) \in \text{QLIN}) \iff (\forall \mathcal{H} \ P(\mathcal{H}) \Rightarrow \mathcal{A}_\beta(\mathcal{H}))$$

This means: any property of the hypergraph of a query grants it quasi-linear decision time if and only if this property implies β -acyclicity.

¹ A hypergraph is said to be k -conformal when every clique of cardinality $\geq k$ is contained in an edge.

Proof. Put ϕ in Disjunctive Normal Form, distribute (existential) quantification over disjunctive clauses which is correct since $\exists x(A(x) \vee B(x)) \Leftrightarrow (\exists xA(x)) \vee (\exists xB(x))$. Each clause is a SCQ whose hypergraph is a subset of $\mathcal{H}(\phi)$, each clause is therefore a β -acyclic SCQ; by Lemma 21, it has a Q_{LN} decision time, therefore so has their disjunction ϕ .

Second point is a direct corollary of this easiness result (part \Leftarrow) and the hardness part of the NCQ dichotomy (part \Rightarrow). \blacktriangleleft

3 Davis-Putnam Resolution with respect to a Nest Point

This section gives an algorithmic result needed for the easiness result; it can be independently read for its own, or may be skipped by admitting it. This part consists in exploiting a particular property of a variable in a CNF formula to perform efficient Davis-Putnam resolution with respect to this variable.

3.1 Definition and Properties

► **Definition 12** (CNF, nest point of a CNF formula, Davis-Putnam resolution). A *CNF formula* $F(x_1, \dots, x_n)$ is a classic propositional formula on the variables x_1, \dots, x_n that is in Conjunctive Normal Form, i.e. $F(x_1, \dots, x_n) = \bigwedge_i C_i$ where C_i are *clauses*, i.e. sub-formulae in the form $C_i = \bigvee_{j=1}^{n(i)} l_i^j$ where l_i^j are *literals*. Literals are either *positive* literals — variables taken in $\{x_1, \dots, x_n\}$ — or *negative* literals — negated variables $\neg x_i$ where $x_i \in \{x_1, \dots, x_n\}$. A formula in CNF can be thought of as a set of clauses, which are sets of literals.

We say a clause C *holds* a variable x when either $x \in C$ or $\neg x \in C$ (or both, but the clause is tautological in this case). The set of variables held by a clause C is denoted $\text{Vars}(C) = \{x_i \mid x_i \in C \text{ or } \neg x_i \in C\}$. We write F_x the set of clauses of F holding a variable x . We say a variable x_i is a *nest point* of $F(x_1, \dots, x_n)$ (see [17]) when for all C_1 and C_2 in F_{x_i} , either $\text{Vars}(C_1) \subseteq \text{Vars}(C_2)$ or $\text{Vars}(C_2) \subseteq \text{Vars}(C_1)$.

We define the resolvent of $F(x_1, \dots, x_n)$ w.r.t. the variable x_1

$$\text{Res}(F, x_1)(x_1, \dots, x_n) = \left\{ C_1 \vee C_2 \mid \begin{array}{l} (C_1 \vee x_1) \in F(x_1, \dots, x_n) \text{ and} \\ (C_2 \vee \neg x_1) \in F(x_1, \dots, x_n) \end{array} \right\}$$

The following results are from [17].

► **Lemma 13** (correctness of resolution, size of resolvent). *The following propositions hold:*

- *the Davis-Putnam resolution is correct:*

$$\forall x_1, \dots, x_n \text{ Res}(F, x_1)(x_1, \dots, x_n) \Leftrightarrow \exists x_1 F_{x_1}(x_1, \dots, x_n)$$

- *if x_1 is a nest point of a formula F , then*

$$\text{Res}(F, x_1) \text{ is a subset of } \{C \setminus \{x_1, \neg x_1\} \mid C \in F_{x_1}\}$$

Proof. Correctness of the resolution is both classical [6] and easy to see.

Let x_1 be a nest point of F . We just have to consider F_{x_1} . Since x_1 is a nest point of F_{x_1} , in F_{x_1} we can rename variables as x_1, \dots, x_n by growing size of the smallest clause containing them. Obviously, $x = x_1$, and all clauses C are such that $\text{Vars}(C) = \{x_1, \dots, x_k\}$ with $k \leq n$. When the computation is over, we just have to give their original names back to the variables.

Take any two clauses $C_1 \vee x$ and $C_2 \vee \neg x$ involved in resolvent computation. We know $\text{Vars}(C_1) \subseteq \text{Vars}(C_2)$ or the converse. Assume, w.l.o.g, $\text{Vars}(C_1) \subseteq \text{Vars}(C_2)$. If some variable is present with different signs in C_1 and C_2 , then the resolvent is tautological. Assume this is not the case. Therefore $C_1 \vee C_2 = C_2$. \blacktriangleleft

3.2 Algorithm and Complexity

Here we prove the resolution can be done in linear time, which is our addition to the main result of Ordyniak et al. ([17]).

► **Lemma 14** (resolvent computation). *Let $F(x_1, \dots, x_n)$ be a CNF sentence whose x_1 is a nest point. We can compute the resolvent of $F(x_1, \dots, x_n)$ w.r.t. x_1 in time $\mathcal{O}(|F|)$.*

Proof. The main algorithmic point consists in adopting a handy representation of the formula. Like in proof of Lemma 13, since x_1 is a nest point of F , we can rename variables in F_{x_1} by growing size of the smallest clause containing it, therefore all clauses C are such that $\text{Vars}(C) = \{x_1, \dots, x_k\}$ with $k \leq n$. Clauses can therefore be encoded as *words* over the alphabet $\{-, +\}$. For example, $x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4$ is encoded as $+ - - +$.

Let us consider subsumed clauses, i.e., clauses C such that we can find C' such that $C' \subseteq C$. With this encoding, C_1 , encoded by w_1 , subsumes C_2 , encoded by w_2 , if and only if w_1 is a prefix of w_2 . Getting rid of subsumed clauses is done by applying the following simple algorithm, where L is the list of words encoding the set of clauses.

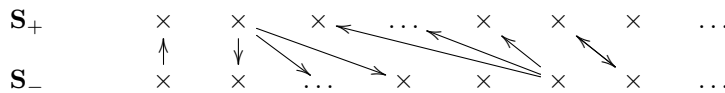
```

RemovePrefixed(L):
  Sort lexicographically L
  n ← 1
  for i from 2 to Card(L) :
    if L[n] not prefix of L[i] :
      L[n + 1] ← L[i]
      n ← n + 1
  L[n + 1] ← End-Of-List
    
```

It is easy to see that this algorithm is correct, i.e. eliminates (in-place) any word that is prefixed by another. Sorting in lexicographical order, here denoted $<$, can be done in linear time using algorithm 3.2 page 80 in [1], see also definition 2. Therefore previous algorithm is linear.

In order to compute the resolvent of F with respect to x_1 , we just need to consider pairs of clauses such that one is in the form $+\dots$ and the other in the form $-\dots$. Let us construct the lexicographically ordered sub-lists S_- and S_+ of words beginning with $-$, resp. $+$, without their leading $-$, resp. $+$. Two clauses C_1 and C_2 respectively encoded by words w_1 and w_2 have a non-tautological resolvent on x_1 if and only if $w_1 \in S_-$ and $w_2 \in S_+$ (or the converse), and w_1 is a prefix of w_2 : in this case, their “resolvent” is w_2 itself, i.e. $C_2 \setminus \{x_1\}$.

Algorithm 1, where S_- (resp. S_+) is represented by a sorted list L_1 (resp. L_2) of n_1 (resp. n_2) elements, is a variant of the fusion algorithm of two sorted lists. It is obviously linear. In order to prove Algorithm 1 is correct, we consider the bipartite directed graph where $a \rightarrow b$ means a is a prefix of b , and where words are represented in growing lexicographical order (from left to right).



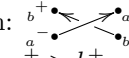
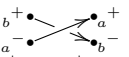
The constraints of this graph are given by the assertions 1-4 given below. As a preliminary, we let the reader convince himself the following fact holds: if a word a is a prefix of a word b , then for all words w and W such that $w < a < W$ and none is a prefix of another, $w < b < W$.


```

ComputeResolvent( $S_-$ ,  $S_+$ ):
   $Res \leftarrow \emptyset$ 
   $i_- \leftarrow 1$ ;  $i_+ \leftarrow 1$ 
  while  $i_- < n_-$  and  $i_+ < n_+$  :
    if  $S_-[i_-]$  is a prefix of  $S_+[i_+]$  :
       $Res \leftarrow Res \cup \{S_+[i_+]\}$ 
      Increment  $i_+$ 
    elseif  $S_+[i_+]$  is a prefix of  $S_-[i_-]$  :
       $Res \leftarrow Res \cup \{S_-[i_-]\}$ 
      Increment  $i_-$ 
    elseif  $S_+[i_+] < S_-[i_-]$  :
      Increment  $i_+$ 
    elseif  $S_+[i_+] > S_-[i_-]$  :
      Increment  $i_-$ 
  return  $Res$ 

```

■ **Algorithm 1** Resolvent computation.

1. The in-degree of the graph is at most one: assuming the contrary leads to $\exists u, v \in S_-$ (resp. S_+) with u prefix of v .
2. The graph has no path of length 2: exactly the same proof.
3. Edges do not cross each other. By symmetry, only two cases have to be considered:
 - Edges do not cross each other in this fashion:  i.e. we can't have a^- prefix of a^+ and b^- prefix of b^+ with $a^- < b^-$ and $a^+ > b^+$. Assume this is the case. a^- is prefix of a^+ and inferior and not prefix of b^- . Therefore $a^+ < b^-$. Since b^- is a prefix of b^+ , $b^- < b^+$, therefore $a^+ < b^+$ (by the preliminary fact), contradictory.
 - Edges don't cross each other in this fashion:  i.e. we can't have a^- prefix of a^+ and b^+ prefix of b^- with $a^- < b^-$ and $a^+ > b^+$: essentially the same proof, but using the preliminary fact twice.
4. The neighbourhood of a vertex is contiguous, i.e. if a^- is a prefix of both b^+ and c^+ , then for all $b^+ < d^+ < c^+$, a^- is a prefix of d^+ .

All these facts together prove that no pair (a, b) where a is a prefix of b is “forgotten” by the given algorithm, which is therefore correct. ◀

4 Easiness Result

In this section, we prove that one can decide a β -acyclic SCQ with n variables on a structure \mathcal{S} in time $\mathcal{O}(n|\mathcal{S}| \log |\mathcal{S}|)$. We prove it a bottom-up fashion: each step either reduces to or generalizes the previous one:

- first step uses results of the first section (Davis-Putnam resolvent computation w.r.t. a nest point in linear time) to decide β -acyclic NCQ-BoolD fast,
- second step generalizes it to β -acyclic SCQ-BoolD, and
- third step reduces β -acyclic SCQ to β -acyclic SCQ-BoolD.

For the sake of simplicity, we always assume a sentence is simple in the following sense: relation symbols all appear once, and in an atom, each variable appears at most once, and in any order of our convenience. This is justified by Corollary 27, page 149.

4.1 NCQ on the Boolean Domain

Here we make use of the results of previous section in order to get the announced complexity.

► **Definition 15** (NCQ-BoolD, SCQ-BoolD). We say a NCQ (resp. SCQ) is *over the boolean domain* when all its domains are fixed to $\{0, 1\}$ instead of being defined with domain symbols; we write it NCQ-BoolD (resp. SCQ-BoolD).

As an example, if $\phi = \exists x_1 \in D_1 \dots x_n \in D_n \psi(x_1, \dots, x_n)$ is a NCQ, then $\phi' = \exists x_1 \dots x_n \in \{0, 1\}^n \psi(x_1, \dots, x_n)$ is a NCQ-BoolD.

The main result of this subsection, Lemma 17, gives an easiness result for NCQ-BoolD having a β -acyclic hypergraph. This result is obtained using inductively the following lemma jointly with the inductive characterisation of β -acyclicity by nest points.

► **Lemma 16** (NCQ-BoolD nest point). *Let ϕ be a NCQ-BoolD, \mathcal{S} a structure, and x a nest point of $\mathcal{H}(\phi)$. We can build another NCQ-BoolD ϕ' and another structure \mathcal{S}' in time $\mathcal{O}(|\mathcal{S}|)$ (independent of $|\phi|$) such that $\text{DecideQ}(\phi)(\mathcal{S}) \Leftrightarrow \text{DecideQ}(\phi')(\mathcal{S}')$ and $\mathcal{H}(\phi') = \mathcal{H}(\phi) \setminus \{x\}$.*

Sketch of proof. Take $\phi = \exists x_1 \dots x_n \in \{0, 1\}^n \psi(x_1, \dots, x_n)$ with

$$\psi(x_1, \dots, x_n) = \bigwedge_i \neg R_i(x_{f(i,1)}, \dots, x_{f(i,n(i))})$$

The main point of the proof is transforming the assertion $\mathcal{S} \models \psi$ into an equivalent CNF formula, and then applying CNF nest point results. For all $(x_1, \dots, x_n) \in \{0, 1\}^n$, we have:

$$\begin{aligned} \mathcal{S} \models \psi(x_1, \dots, x_n) &\Leftrightarrow \bigwedge_i \neg R_i^{\mathcal{S}}(x_{f(i,1)}, \dots, x_{f(i,n(i))}) \\ &\Leftrightarrow \bigwedge_i \bigwedge_{(a_1, \dots, a_{n(i)}) \in R_i^{\mathcal{S}}} \underbrace{(x_{f(i,1)}, \dots, x_{f(i,n(i))}) \neq (a_1, \dots, a_{n(i)})}_{C_i(a_1, \dots, a_{n(i)})} \\ C_i(a_1, \dots, a_{n(i)}) &\Leftrightarrow (x_{f(i,1)} \neq a_1) \vee \dots \vee (x_{f(i,n(i))} \neq a_{n(i)}) \\ &\Leftrightarrow \sigma(a_1)x_{f(i,1)} \vee \dots \vee \sigma(a_{n(i)})x_{f(i,n(i))} \end{aligned}$$

where σ maps 1 to \neg and 0 to ϵ . Finally, for all x_1, \dots, x_n we have the equivalence:

$$\mathcal{S} \models \psi(x_1, \dots, x_n) \Leftrightarrow \underbrace{\bigwedge_i \bigwedge_{(a_1, \dots, a_{n(i)}) \in R_i^{\mathcal{S}}} \sigma(a_1)x_{f(i,1)} \vee \dots \vee \sigma(a_{n(i)})x_{f(i,n(i))}}_{F(x_1, \dots, x_n)}$$

Clearly, x is a nest point of $F(x_1, \dots, x_n)$ (see the note at the end of previous definition). The transformation $(\phi, \mathcal{S}) \mapsto F$ is done in time $\mathcal{O}(|\mathcal{S}|)$ — i.e. linear time.² It is rather easy to see that the reverse transformation (from the propositional formula back to the corresponding NCQ-BoolD) can be done in linear time.

Lemma 13 states that $\text{Res}(F, x)$ is a subset of F_x where x was removed, and that Davis-Putnam resolution is correct: $\forall x_1, \dots, x_n \text{ Res}(F, x)(x_1, \dots, x_n) \Leftrightarrow \exists x F_x(x_1, \dots, x_n)$. Notice $x \in \{x_1, \dots, x_n\}$. Furthermore, Lemma 14 states this can be done in linear time.

² Not exactly: the variables of the CNF are “bigger” than the 0/1 present in the structure. Nevertheless, when applying resolution, they will be encoded as signs — i.e. either + or -. In fact, the CNF form *explains* correctness but is not an actual step.

The following algorithm uses previous notations and is justified by the arguments above.

```

RemoveNestPoint( $\phi, x, \mathcal{S}$ ):
  //  $\psi$  is in the following form:  $\bigwedge_i \neg R_i(x_{f(i,1)}, \dots, x_{f(i,n(i))})$ 
  // Recall that  $\sigma(1) = \neg$ ,  $\sigma(0) = \epsilon$ 
   $F \leftarrow \bigwedge_i \bigwedge_{(a_1, \dots, a_{n(i)}) \in R_i^S} \sigma(a_1)x_{f(i,1)} \vee \dots \vee \sigma(a_{n(i)})x_{f(i,n(i))}$ 
  Replace  $\neg R(x_1, \dots, x_n, x)$  by  $\neg R(x_1, \dots, x_n)$  in  $\phi$ 
   $F \leftarrow \text{Res}(F, x)$ 
  Rebuild  $\mathcal{S}$  from  $F$ 
  return  $(\phi, \mathcal{S})$ 

```

This algorithm clearly uses linear time, i.e. $\mathcal{O}(|\mathcal{S}|)$. ◀

We now state the NCQ easiness result. This lemma is stated and proved in order to give a simplified view of Lemma 19 below.

► **Lemma 17** (NCQ-BoolD easiness result). *A NCQ-BoolD ϕ with n variables such that $\mathcal{H}(\phi)$ is β -acyclic can be decided in time $\mathcal{O}(n|\mathcal{S}|)$.*

Proof. The following does it:

```

DecideNCQ – BoolD( $\phi$ )( $\mathcal{S}$ ):
  We know  $(x_1, \dots, x_n)$  is a REO of  $\phi$ .
  for  $i$  from  $n$  to 1 :
     $(\phi, \mathcal{S}) \leftarrow \text{RemoveNestPoint}(\phi, x_i, \mathcal{S})$ 
    while  $\phi$  contains some negative conjunct  $\neg R()$  :
      if  $R^S \neq \emptyset$  : return False
      Remove  $\neg R()$  from  $\phi$ 
  return True

```

Let us justify its correctness. By previous β -acyclicity characterisation, we know we can apply nest point removing until there is no variable left.

While nest points are inductively removed, some relations become of arity 0. The interpretation of these relations R is either empty — in this case, the assertion $\neg R()$ is a tautology and can be removed from the conjunction — or non-empty, i.e. contains only the empty tuple. In this case, the assertion $\neg R()$ is a contradiction, and the query should return “false” to mean this conjunction is not satisfiable (i.e. the sentence is not satisfied).

Since resolvent computation is done in linear time, each loop turn takes linear time, which happens n times. ◀

4.2 Easiness Result for SCQ on the Boolean Domain

We now refine previous result, instead of using it. It is barely more complicated.

Instead of proving easiness of general β -acyclic NCQ, we directly treat the case of SCQ. A first reason is that domains will be positive relations, therefore a NCQ is already a kind of SCQ; treating directly SCQ avoids treating domains specifically. The other reason is discussed after the following lemma.

This trick is a (weak) variant of the one presented in [19], and was inspired by it. It looks simple but is the key point allowing extension of the NCQ easiness result to wider classes.

► **Lemma 18.** *Let R_a^+ be a boolean relation of arity a . We can build in linear time the boolean relations R_{a-1}^+ of arity $a - 1$ and R_a^- of arity a such that, for all $x_1, \dots, x_a \in \{0, 1\}^a$:*

- $R_a^+(x_1, \dots, x_a) \Leftrightarrow R_{a-1}^+(x_1, \dots, x_{a-1}) \wedge \neg R_a^-(x_1, \dots, x_a)$
- $(x_1, \dots, x_a) \in R_a^- \Rightarrow (x_1, \dots, x_{a-1}) \in R_{a-1}^+$

```

DecideSCQ – BoolD( $\phi$ )( $\mathcal{S}$ ):
┌ We know  $(x_1, \dots, x_n)$  is a REO of  $\phi$ .
├ for  $i$  from  $n$  to  $1$  :
│   ┌ while there is more than one positive conjunct containing  $x_i$  in  $\phi$  :
│   │   ┌ Take  $R(\bar{y}, \bar{z}, x_i)$  and  $S(\bar{z}, x_i)$  among them
│   │   │   ┌  $R^S \leftarrow \{(\bar{a}, \bar{b}, c) \in R^S \mid (\bar{b}, c) \in S^S\}$ 
│   │   │   └ Remove  $S(\bar{z}, x_i)$  from  $\phi$ 
│   │   └ if  $x_i$  is contained in a positive conjunct  $R_a^+(\bar{y}, x)$  :
│   │       ┌ Build  $(R_{a-1}^+)^S, (R_a^-)^S$  from  $(R_a^+)^S$ 
│   │       │   ┌ Replace  $R_a^+(\bar{y}, x_i)$  by  $R_{a-1}^+(\bar{y}) \wedge \neg R_a^-(\bar{y}, x_i)$  in  $\phi$ 
│   │       │   └  $(\phi, \mathcal{S}) \leftarrow \text{RemoveNestPoint}(\phi, x_i, \mathcal{S})$ 
│   │       │       ┌  $(R_{a-1}^+)^S \leftarrow (R_{a-1}^+)^S \setminus (R_a^-)^S$ 
│   │       └ else
│   │           ┌  $(\phi, \mathcal{S}) \leftarrow \text{RemoveNestPoint}(\phi, x_i, \mathcal{S})$ 
│   │           └ while  $\phi$  contains a negative (resp. positive) conjunct  $\neg R()$  (resp.  $R()$ ) :
│   │               ┌ if  $R^S \neq \emptyset$  (resp.  $R^S = \emptyset$ ) :
│   │               │   ┌ return False
│   │               └ Remove  $\neg R()$  (resp.  $R()$ ) from  $\phi$ 
│   └ return True

```

■ **Algorithm 2** SCQ-BoolD decision algorithm.

Proof. Let $R_{a-1}^+ = \{(e_1, \dots, e_{a-1}) \mid \exists e_a R_a^+(e_1, \dots, e_a)\}$ and $R_a^- = (R_{a-1}^+ \times \{0, 1\}) \setminus R_a^+$. ◀

We could transform a SCQ-BoolD into a NCQ-BoolD, at the cost of an additional n factor, that would affect the complexity of general SCQ. However, by making a somewhat *ad hoc* algorithm, we can treat the SCQ-BoolD case *directly* and get the expected complexity $\mathcal{O}(n|\mathcal{S}|)$.

► **Lemma 19** (SCQ-BoolD easiness result). *A SCQ-BoolD ϕ with n variables, and whose hypergraph $\mathcal{H}(\phi)$ is β -acyclic can be decided in time $\mathcal{O}(n|\mathcal{S}|)$.*

Proof. Notice the statements of the proof of Lemma 17 still hold. Apply Algorithm 2. If several positive conjuncts hold a given nest point, then we can proceed to filtering as follows: take any two positive conjuncts. The set of variables held by one includes the set of variables held by the other, we can sort them in a lexicographical order such that the set of shared variables have the strong weight of the lexicographical order; we can finally proceed to filtering in linear time. β -acyclicity is preserved, and the REO is maintained by edge removal. In the end, there is only one positive conjunct holding the nest point.

Now the positive conjunct can be managed with Lemma 18, summed up in the following, where R/a means the relation R has arity a .

$$(R_a^+)^S/a \xrightarrow{\text{Lemma 18}} (R_{a-1}^+)^S/a-1 \xrightarrow{\text{obvious}} (R_{a-1}^+)^S/a-1$$

$$(R_a^+)^S/a \xrightarrow{\text{Rem.NestP.}} (R_a^-)^S/a \xrightarrow{\text{Rem.NestP.}} (R_a^-)^S/a-1 \xrightarrow{\text{obvious}} (R_{a-1}^+)^S/a-1$$

We build $(R_{a-1}^+)^S, (R_a^-)^S$ from $(R_a^+)^S$ in linear time. After linear time resolvent computation, $(R_a^-)^S$ has arity $a-1$ (the nest point has been removed) and we can proceed to a simplification in linear time justified by:

$$(R_{a-1}^+)^S(x_1, \dots, x_{a-1}) \wedge \neg (R_a^-)^S(x_1, \dots, x_{a-1}) \Leftrightarrow ((R_{a-1}^+)^S \setminus (R_a^-)^S)(x_1, \dots, x_{a-1})$$

Since resolvent computation is done in linear time, each loop turn takes linear time, which happens n times. ◀

4.3 Final Easiness Result

Here is the last (technical) result on hypergraphs we need.

► **Lemma 20.** *Let \mathcal{H}_1 , $x \in \mathcal{V}(\mathcal{H}_1)$, $y \notin \mathcal{V}(\mathcal{H}_1)$ and $\mathcal{H}_2 = \{e \cup \{x, y\} \mid e \cup \{x\} \in \mathcal{H}_1\} \cup \{e \in \mathcal{H}_1 \mid x \notin e\}$. This means \mathcal{H}_2 is the same hypergraph as \mathcal{H}_1 except every edge holding x also holds y .*

We have: $(a_1, \dots, a_n, x, b_1, \dots, b_m)$ is a REO of \mathcal{H}_1 iff $(a_1, \dots, a_n, x, y, b_1, \dots, b_m)$ and $(a_1, \dots, a_n, y, x, b_1, \dots, b_m)$ are REOs of \mathcal{H}_2 . In particular, $\mathcal{A}_\beta(\mathcal{H}_1) \Leftrightarrow \mathcal{A}_\beta(\mathcal{H}_2)$.

Proof. Easy considering the inductive definition of β -acyclicity (definition 7). ◀

► **Lemma 21 (SCQ easiness result).** *A SCQ ϕ with n variables, and whose hypergraph $\mathcal{H}(\phi)$ is β -acyclic can be decided in time $\mathcal{O}(n|\mathcal{S}| \log |\mathcal{S}|)$.*

Proof. We transform a β -acyclic SCQ with n variables into a β -acyclic SCQ-BoolD having $n \log |\mathcal{S}|$ variables. Let ϕ be a β -acyclic SCQ admitting (x_1, \dots, x_n) as a REO. Finding it takes a time *depending only on ϕ* .

We assume domains are reasonably encoded: we suppose there is a constant k such that, for any domain D_i , $\lceil \log_2 \max_i (D_i^S) \rceil < k \log |\mathcal{S}|$. This assumption is reasonable: we always can sort (in linear time) the union of the domains and associate each element with its number in this order. Then we can re-encode (in-place) the whole structure in time $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$ by, for each occurring element, looking for its number, and replacing it by its number. Each element has therefore a size $\mathcal{O}(\log |\mathcal{S}|)$.

Let $s_i = \lceil \log_2 \max (D_i^S) \rceil$. By the previous point, $s_i = \mathcal{O}(\log |\mathcal{S}|)$. We define

$$\phi' = \exists x_1 \in \{1, \dots, 2^{s_1}\} \dots \exists x_n \in \{1, \dots, 2^{s_n}\} \quad D_1(x_1) \wedge \dots \wedge D_n(x_n) \wedge \psi$$

We have $\mathcal{H}(\phi') = \mathcal{H}(\phi) \cup \{\{x_i\} \mid x_i \in \mathcal{V}(\mathcal{H}(\phi))\}$ therefore $\mathcal{H}(\phi')$ is β -acyclic. Now we can transform ϕ' into a SCQ-BoolD:

$$\phi'' = \exists (x_1^1, \dots, x_1^{s_1}, \dots, x_n^1, \dots, x_n^{s_n}) \in \{0, 1\}^{\sum_{i=1}^n s_i} \quad D_1(x_1^1, \dots, x_1^{s_1}) \wedge \dots \wedge D_n(x_n^1, \dots, x_n^{s_n}) \wedge \psi'$$

with ψ' the same as ψ with every $R(x_a, x_b, \dots)$ replaced by $R(x_a^1, \dots, x_a^{s_a}, x_b^1, \dots, x_b^{s_b}, \dots)$. ϕ'' is a SCQ-BoolD, also β -acyclic due to Lemma 20. Moreover, $(x_1^1, \dots, x_1^{s_1}, \dots, x_n^1, \dots, x_n^{s_n})$ is a REO of $\mathcal{H}(\phi'')$. Noting $\sum_{i=1}^n s_i = \mathcal{O}(n \log |\mathcal{S}|)$ and applying Lemma 19 concludes. ◀

5 Hardness Result

We introduce a basic notion of reduction together with a trivial lemma that will be useful to prove Corollary 27, and makes simple the proof of the hardness result.

► **Definition 22 (linear reduction \prec).** We say a problem P_1 *reduces linearly* to a problem P_2 , denoted $P_1 \prec P_2$, when we can find $f \in \text{LIN}$ such that, given an algorithm A_2 deciding P_2 , $A_2 \circ f$ decides P_1 that is to say the following algorithm decides P_1 :

```
Decide $P_1(I_1)$ :
┌    $I_2 \leftarrow f(I_1)$ 
└   return Decide $P_2(I_2)$ 
```

We also say P_2 *expresses* P_1 . When P_1 both reduces linearly to and expresses linearly P_2 , we say P_1 and P_2 are *equivalent*, denoted $P_1 \sim P_2$.

The following lemma is obvious:

► **Lemma 23** (linear reduction properties). *Linear reduction is transitive, i.e. if $P_1 \prec P_2$ and $P_2 \prec P_3$, then $P_1 \prec P_3$; and QLIN is closed by linear reduction, i.e. if $P_1 \prec P_2$ and $P_2 \in \text{QLIN}$ then $P_1 \in \text{QLIN}$.*

5.1 A Technical Point

In order to prove our hardness result on NCQ, we need to introduce a normalized form of NCQ; it is used to show that the complexity of a NCQ is exactly determined by its hypergraph.

► **Definition 24** (simple, sorted, normalized NCQ). One says that a NCQ $\phi = \exists x_1 \in D_1 \dots \exists x_n \in D_n \psi$ is *simple* if each relation symbol occurs only once in ψ . ϕ is *sorted* if the tuple of variables of each atom of ψ occurs in increasing order of subscripts with no repetition, i.e. every atom of ψ is in the form $R(x_{i_1}, \dots, x_{i_k})$ where $i_1 < \dots < i_k$.

A NCQ is *normalized* if it is both simple and sorted.

The following lemma is obvious:

► **Lemma 25.** *Let ϕ_1, ϕ_2 be two normalized NCQ with the same signature and $\mathcal{H}(\phi_1) = \mathcal{H}(\phi_2)$. Then ϕ_1 and ϕ_2 are identical up to relation symbols permutation. In particular $\text{DecideQ}(\phi_1) \sim \text{DecideQ}(\phi_2)$.*

► **Lemma 26.** *Any NCQ ϕ is equivalent to some normalized NCQ ϕ'' with $\mathcal{H}(\phi) = \mathcal{H}(\phi'')$, i.e. $\text{DecideQ}(\phi) \sim \text{DecideQ}(\phi'')$.*

Sketch of proof. Here we give the only non-trivial point of the proof, in an easy to generalize example. Let $\phi_1 = \exists x_1 \in D_1 \exists x_2 \in D_2 \exists x_3 \in D_3 \neg R_1(x_1, x_2) \wedge \neg R_2(x_2, x_3)$ and ϕ_2 built by replacing *both* R_1 and R_2 by the same symbol R in ϕ_1 . We prove $\text{DecideQ}(\phi_1) \prec \text{DecideQ}(\phi_2)$. We define, for $i \in \{1, 2, 3\}$, $D_i^{\mathcal{S}_2} = D_i^{\mathcal{S}_1} \times \{i\}$ and $R^{\mathcal{S}_2} = \{((a_1, 1), (a_2, 2)) \mid (a_1, a_2) \in R_1^{\mathcal{S}_1}\} \cup \{((a_2, 2), (a_3, 3)) \mid (a_2, a_3) \in R_2^{\mathcal{S}_1}\}$. The key point is that $R^{\mathcal{S}_2}$ is a *disjoint* union of relations corresponding to relations $R_1^{\mathcal{S}_1}$ and $R_2^{\mathcal{S}_1}$; further, the tuples originating from $R_1^{\mathcal{S}_1}$ (resp. $R_2^{\mathcal{S}_1}$) are *identified* by their specific form $((a_1, 1), (a_2, 2))$ (resp. $((a_2, 2), (a_3, 3))$). ◀

► **Corollary 27.** *For all NCQ ϕ_1 and ϕ_2 , we have*

$$\mathcal{H}(\phi_1) = \mathcal{H}(\phi_2) \quad \Rightarrow \quad \text{DecideQ}(\phi_1) \sim \text{DecideQ}(\phi_2)$$

Proof. Obvious corollary of Lemma 25 and Lemma 26. ◀

5.2 Hardness Result

► **Lemma 28.** *Let ϕ be a NCQ, and x a variable appearing in ϕ . Let ϕ' be the query obtained by removing every occurrence of x in ϕ , that is to say removing the $\exists x \in D_x$, and removing x in atoms where it occurs. Then $\text{DecideQ}(\phi)$ expresses linearly $\text{DecideQ}(\phi')$.*

Proof. By virtue of Corollary 27, we can assume that every relation appears once in ϕ , with associated variables in order. For simplicity, x is therefore assumed the minimal element for the order. To define the reduction, we will just define how a given relation R is transposed between \mathcal{S}_1 and \mathcal{S}_2 . If x does not appear in the corresponding atom, there is no difference i.e. $R^{\mathcal{S}_2} = R^{\mathcal{S}_1}$.

In the other case, R appears as $R(x, x_{a(1)}, \dots, x_{a(k)})$ in ϕ , with $a : \{1, \dots, k\} \rightarrow \{1, \dots, n\}$. R appears as $R(x_{a(1)}, \dots, x_{a(k)})$ in ϕ' . From now, completing the reduction is easy: define $R^{S^2} = \{(1, s_1, \dots, s_k) \mid (s_1, \dots, s_k) \in R^{S^1}\}$ and $D_x^{S^2} = \{1\}$. ◀

► **Corollary 29.** *Let $\phi \in \text{NCQ}$. For every ϕ' such that $\mathcal{H}(\phi') = \mathcal{H}(\phi)[S]$ for some S , we have $\text{DecideQ}(\phi) \succ \text{DecideQ}(\phi')$.*

Proof. For every $x \in \mathcal{V}(\mathcal{H}(\phi)) \setminus S$, apply Lemma 28, together with transitivity (Lemma 23). We have proved that $\text{DecideQ}(\phi)$ expresses *some* $\text{DecideQ}(\phi')$, such that $\mathcal{H}(\phi') = \mathcal{H}(\phi)[S]$; apply Corollary 27 to prove equivalence of this last query with *all* queries having the same hypergraph; transitivity concludes. ◀

► **Lemma 30 (hardness result).** *Under hypothesis that the problem of deciding the presence of a triangle in a graph on n vertices cannot be decided in $\mathcal{O}(n^2 \log n)$:*

$$\forall \phi \in \text{NCQ} \quad \text{DecideQ}(\phi) \in \text{qLIN} \Rightarrow \mathcal{A}_\beta(\mathcal{H}(\phi))$$

Proof. For the sake of contradiction, assume $\text{DecideQ}(\phi) \in \text{qLIN}$ and $\neg \mathcal{A}_\beta(\mathcal{H}(\phi))$. This implies we can find $S \subseteq \mathcal{V}(\mathcal{H})$ and $h \subseteq \mathcal{H}(\phi)$ such that $h[S]$ is a chordless cycle. Then, by Corollary 29 and Corollary 27, $\text{DecideQ}(\phi)$ expresses the following query of signature σ :

$$P = \text{DecideQ} \left(\exists x_1 \in D_1 \dots \exists x_k \in D_k \neg R_k(x_k, x_1) \wedge \bigwedge_{i=1}^{k-1} \neg R_i(x_i, x_{i+1}) \right)$$

that, by Lemma 23, is also in qLIN .

Now, let us associate to any graph $G = (V, E)$ with $\text{Card}(V) = n$ a σ -structure defined as follows. For each R_i with $i > 3$, set $R_i^S = \{(i, j) \in V^2 \mid i \neq j\}$. For each R_i with $i \leq 3$, set $R_i^S = \{(i, j) \in V^2 \mid (i, j) \notin E\}$. Set $D_i^S = V$. We have $|\mathcal{S}| = \mathcal{O}(|V|^2) = \mathcal{O}(n^2)$. If this query is in qLIN , we can decide the presence of a triangle in G in time $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|) = \mathcal{O}(n^2 \log n^2) = \mathcal{O}(n^2 \log n)$. ◀

Concluding Remark

β -acyclic existential first-order queries have many qualities, they only lack one thing: to include α -acyclic CQ. This is to be addressed in a future paper.

Acknowledgments

The author expresses his gratitude to Étienne Grandjean for careful, multiple readings of the successive versions of this paper, despite his busy schedule, until any doubtful or unclear point had disappeared. Without his assistance, this paper would still be a confused set of sketchy notes.

The author would also like to thank anonymous referees for their quite detailed reviews that greatly contributed to improve the quality of the paper.

References

- 1 Aho, Hopcroft, and Ullman. *The design and analysis of computer algorithms*. 1974.
- 2 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. *Computer Science Logic*, 4646:208–222, 2007.

- 3 Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.
- 4 Andries E. Brouwer and Antoon W. J. Kolen. A super-balanced hypergraph has a nest point. *Technical report, Math. centr. report ZW146, Amsterdam*, 1980.
- 5 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. *ACM Symp. on Theory of Computing*, pages 77–90, 1977.
- 6 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.
- 7 Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artif. Intell.*, 38(3):353–366, 1989.
- 8 David Duris. Some characterizations of gamma and beta-acyclicity of hypergraphs. November 2008.
- 9 Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30:514–550, 1983.
- 10 Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.
- 11 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.
- 12 Etienne Grandjean. Sorting, Linear Time and the Satisfiability Problem. *Ann. Math. Artif. Intell.*, 16:183–236, 1996.
- 13 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 657–666, 2001.
- 14 Yuri Gurevich and Saharon Shelah. Nearly linear time. In Albert Meyer and Michael Taitlin, editors, *Logic at Botik '89*, volume 363 of *Lecture Notes in Computer Science*, pages 108–118. Springer Berlin / Heidelberg, 1989.
- 15 Phokion G. Kolaitis. Constraint satisfaction, databases, and logic. In *IJCAI*, pages 1587–1595, 2003.
- 16 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.
- 17 Sebastian Ordyniak, Daniel Paulusma, and Stefan Szeider. Satisfiability of Acyclic and Almost Acyclic CNF Formulas. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 84–95, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- 18 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings Conf. on Very Large Databases*, pages 82–94, 1981.
- 19 Bruno Zanuttini and Jean-Jacques Hébrard. A unified framework for structure identification. *Information Processing Letters*, 81(6):335 – 339, 2002.