Direct Access for Conjunctive Queries with Negations

Florent Capelli, Oliver Irwin

Direct Access for Conjunctive Queries with Negations

Florent Capelli
Univ. Artois, CNRS, UMR 8188, Centre de Recherche en Informatique de Lens (CRIL), F-62300 Lens, France

Oliver Irwin
Université de Lille, CNRS, Inria, UMR 9189 - CRIStAL, F-59000 Lille, France

Abstract

Given a conjunctive query Q and a database $\mathbf{D}$, a direct access to the answers of Q over $\mathbf{D}$ is the operation of returning, given an index j, the j^{th} answer for some order on its answers. While this problem is #P-hard in general with respect to combined complexity, many conjunctive queries have an underlying structure that allows for a direct access to their answers for some lexicographical ordering that takes polylogarithmic time in the size of the database after a polynomial time precomputation. Previous work has precisely characterised the tractable classes and given fine-grained lower bounds on the precomputation time needed depending on the structure of the query. In this paper, we generalise these tractability results to the case of signed conjunctive queries, that is, conjunctive queries that may contain negative atoms. Our technique is based on a class of circuits that can represent relational data. We first show that this class supports tractable direct access after a polynomial time preprocessing. We then give bounds on the size of the circuit needed to represent the answer set of signed conjunctive queries depending on their structure. Both results combined together allow us to prove the tractability of direct access for a large class of conjunctive queries. On the one hand, we recover the known tractable classes from the literature in the case of positive conjunctive queries. On the other hand, we generalise and unify known tractability results about negative conjunctive queries – that is, queries having only negated atoms. In particular, we show that the class of β-acyclic negative conjunctive queries and the class of bounded nest set width negative conjunctive queries admit tractable direct access.

2012 ACM Subject Classification Information systems → Relational database model

Keywords and phrases Conjunctive queries, factorized databases, direct access, hypertree decomposition

Digital Object Identifier 10.4230/LIPIcs.ICDT.2024.13

Related Version *Full Version*: `https://arxiv.org/abs/2310.15800` [10]

Funding This work was supported by project ANR KCODA, ANR-20-CE48-0004.

Acknowledgements We are thankful to Stefan Mengel and Sylvain Salvati for helpful discussions while elaborating these results.

1 Introduction

The *direct access (DA for short) task* is the problem of outputing, given k, the k-th answer of a query Q over a database $\mathbf{D}$. An error is returned if k is greater than the number of answers of Q. An order on $[\![Q]\!]^{\mathbf{D}}$, the answers of Q over $\mathbf{D}$, is assumed. This task has been introduced by Bagan, Durand, Grandjean and Olive in [2] and is very natural in the context of databases. It can be used as a building block for many other interesting tasks such as counting, enumerating [2] or sampling without repetition [13, 22] the answers of Q. Of course, if one has access to an ordered array containing $[\![Q]\!]^{\mathbf{D}}$, answering DA tasks simply consists in reading the right entry of the array. However, building such an array is often expensive,

especially when the number of answers of Q is large. Hence, a natural approach for solving this problem is to simulate this method by using a data structure to represent $[\![Q]\!]^{\mathbf{D}}$ that still allows for efficient DA tasks to be solved but that is cheaper to compute than the complete answer set. Moreover, one is rarely interested in accessing exactly one tuple of the answer set but one is usually interested in having a data structure allowing to solve efficiently DA tasks for any input k. Hence the time needed to solve exactly one DA task is not the best measure of complexity. To compare algorithms for DA tasks, it is then relevant to allow a *preprocessing phase* to construct a data structure that can be used later during the *access phase* where one needs to efficiently answer any DA task. Hence, we say that a method solves the DA problem if it consists in two algorithms P and A such that: on input Q and $\mathbf{D}$, P construct a data structure S and $A(S, k)$ returns the k-th answer of Q for any k. To measure the quality of such an algorithm, we hence separate the *preprocessing time* – that is the time needed for executing $P(Q, \mathbf{D})$ – and the *access time*, that is, the time needed to execute $A(S, k)$. For example, the method consisting in building an indexed array for $[\![Q]\!]^{\mathbf{D}}$ solves DA with preprocessing time at least equal to the size of $[\![Q]\!]^{\mathbf{D}}$ (and much higher in practice) and constant access time. While the access time is optimal in this case, the cost of preprocessing is often too high to pay in practice.

Previous work has consequently focused on devising methods with better preprocessing time while offering reasonable access time. In their seminal work [2], Bagan, Durand, Grandjean and Olive give a method for solving the DA problem with linear precomputation time and constant access time on a that works on the class of first order logic formulas and bounded degree databases. Bagan [1] later studied the problem for monadic second order formulas over bounded treewidth databases. Another line of research has focused on classes of conjunctive queries that admit efficient method for solving the DA problem. In [13], Carmeli, Zeevi, Berkholz, Kimelfeld, and Schweikardt give a method solving the DA problem for acyclic conjunctive queries with linear preprocessing time and polylogarithmic access time for a well-chosen lexicographical order. The results also hold for bounded fractional hypertree width queries, a number measuring how far a conjunctive query is from being acyclic. It generalises many results from the seminal paper by Yannakakis establishing the tractability of testing non-emptiness of acyclic conjunctive queries [33] to the tractability of counting the number of answers of conjunctive queries [30] having bounded hypertree width. later improved this result by precisely characterising the lexicographical ordering allowing for this kind of complexity guarantees. Fine-grained characterisation of the complexity of solving the DA problem on conjunctive queries, whose answers are assumed to be ordered by some lexicographical order, has been given by Carmeli, Tziavelis, Gatterbauer, Kimelfeld and Riedewald in [12] for the case of acyclic queries and by Bringmann, Carmeli and Mengel in [7] for the general case. Recently, Eldar, Carmeli and Kimelfeld [15] studied the complexity of solving the DA problem for conjunctive queries with aggregation.

In this paper, we devise new methods for solving the DA problem for *signed conjunctive queries*, that is, conjunctive queries that may contain negated atoms. This is particularly challenging because only a few tractability results are known on signed conjunctive queries. The problem of testing non-emptiness of signed conjunctive queries being NP-hard on acyclic conjunctive queries with respect to combined complexity, it is not possible to directly build on the work cited in the last paragraph. Two classes of negative conjunctive queries (that is, conjunctive queries where every atom is negated) have been shown so far to support efficient non-emptiness testing: the class of β-acyclic queries [29, 4] and the class of bounded nested-set width queries [25]. The former has been shown to also support efficient (weighted) counting [6, 9]. Our main contribution is a generalisation of these results to DA. More

precisely, we give a method that efficiently solves the DA problem on a large class of signed conjunctive queries, which contains in particular β-acyclic negative conjunctive queries, bounded nest-width negative conjunctive queries and bounded fractional hypertree width positive conjunctive queries. For the latter case, the complexity we obtain is similar to the one presented in [7] and we also get complexity guarantees depending on a lexicographical ordering that can be specified by the user. Hence our result both improves the understanding of the tractability of signed conjunctive queries and unifies the existing results with the positive case. In a nutshell, we prove that the complexity of solving the DA problem for a lexicographical order of a signed conjunctive query Q roughly matches the complexity proven in [7] for the worst positive query we could construct by removing some negative atoms of Q and turning the remaining ones to positive atoms. It is not surprising that the complexity of solving the DA problem on signed conjunctive queries depends on the complexity of solving the DA problem for subqueries since one could simulate direct access to such a subquery by choosing a database where the removed negated atoms are associated with empty relations, hence, making them virtually useless in the query. However, proving the actual combined complexity upper bound is not trivial to obtain and necessitates introducing new tools to handle negated atoms.

As a byproduct, we introduce a new notion of hypergraph width based on elimination order, the β-hyperorder width, that is hereditary – in the sense that the width of every subhypergraph does not exceed the width of the original hypergraph – which makes it particularly well tailored for studying the tractability of negative conjunctive queries. We show that this notion sits between nest-set width and β-hypertree width [18], but do not suffer from one important drawback of working with β-hypertree width: our width notion is based on a decomposition that works for every subhypergraph.

Our method is based on a two-step preprocessing. Given a signed conjunctive query Q, a database $\mathbf{D}$ and an order $\prec$ on its variables, we start by constructing a circuit which computes $[\![Q]\!]^{\mathbf{D}}$ in a factorised way enjoying interesting syntactical properties. The size of this circuit depends on the complexity of the order $\prec$ chosen on the variables of Q. We then show that, with a second light preprocessing on the circuit itself, we can answer DA tasks for the lexicographical order induced by $\prec$ on the circuit in time $\mathsf{poly}(n)\mathsf{polylog}(D)$ where n is the number of variables of Q and D is the domain of $\mathbf{D}$. This approach is akin to the approach used in *factorised databases* introduced by Olteanu and Závodný [27], a fruitful approach allowing efficient management of the answer set of a query by working directly on a factorised representation of the answer set instead of working on the query itself [26, 32, 3, 28]. However, the restrictions that we are considering in this paper are different from the one used in previous work since we need somehow to account for the variable ordering in the circuit itself. The syntactic restrictions we use have already been considered in [9] where they are useful to deal with β-acyclic CNF formulas.

The paper is organised as follows: Section 2 introduces the notations and concepts necessary to understand the paper. We then present the family of circuits we use to represent database relations and the DA method for circuits in Section 3. Section 4 presents the algorithm used to construct a circuit representing $[\![Q]\!]^{\mathbf{D}}$ from a join query Q (that is a conjunctive query without existential quantifiers) and a database $\mathbf{D}$. Upper bounds on the size of the circuits are given in Section 4.3 using hypergraph decompositions defined in Section 4.2. Finally Section 5 explicitly states the results we obtain by combining both techniques together, explains how one can go from join query to conjunctive query by existentially projecting variables directly in the circuit and makes connections with the existing literature. We conclude with interesting research directions in Section 6.

Due to space restriction, full proofs have been omitted from the paper. Proofs of theorems and lemmas tagged with a star symbol $\star$ can be found in the full version of this paper [10].

2 Preliminaries

Given $n \in \mathbb{N}$, we denote by $[n]$ the set $\{0, \dots, n\}$. When writing down complexity, we use the notation $\mathsf{poly}(n)$ to denote that the complexity is polynomial in n, $\mathsf{poly}_k(n)$ to denote that the complexity is polynomial in n when k is considered a constant (in other words, the coefficients and the degree of the polynomial may depend on k) and $\mathsf{polylog}(n)$ to denote that the complexity is polynomial in $\mathsf{log}(n)$.

Tuples and relations. Let D and X be finite sets. A (named) *tuple* on domain D and variables X is a mapping from X to D. We denote by D^X the set of all tuples on domain D and variables X. A *relation R on domain D and variables X* is a subset of tuples, that is, $R \subseteq D^X$. Given a tuple $\tau \in D^X$ and $Y \subseteq X$, we denote by $\tau|_Y$ the tuple on domain D and variable Y such that $\tau|_Y(y) = \tau(y)$ for every $y \in Y$. Given a variable $x \in X$ and $d \in D$, we denote by $[x \leftarrow d]$ the tuple on variables $\{x\}$ that assigns the value $d \in D$ to x. We denote by $\langle\rangle$ the empty tuple, that is, the only element of $D^\emptyset$. Given two tuples $\tau_1 \in D^{X_1}$ and $\tau_2 \in D^{X_2}$, we say that τ_1 and τ_2 are *compatible*, denoted by $\tau_1 \simeq \tau_2$, if $\tau_1|_{X_1 \cap X_2} = \tau_2|_{X_1 \cap X_2}$. In this case, we write $\tau_1 \bowtie \tau_2$ the tuple on domain D and variables $X_1 \cup X_2$ defined as $(\tau_1 \bowtie \tau_2)(x)$ to be $\tau_1(x)$ if $x \in X_1$ and $\tau_2(x)$ otherwise. If $X_1 \cap X_2 = \emptyset$, we write $\tau_1 \times \tau_2$. The *join $R_1 \bowtie R_2$ of R_1 and R_2*, for two relations R_1, R_2 on domain D and variables X_1, X_2 respectively, is defined as $\{\tau_1 \bowtie \tau_2 \mid \tau_1 \in R_1, \tau_2 \in R_2, \tau_1 \simeq \tau_2\}$. Observe that if $X_1 \cap X_2 = \emptyset$, $R_1 \bowtie R_2$ is simply the *Cartesian product of R_1 and R_2*. Then, we denote it by $R_1 \times R_2$.

Let $R \subseteq D^X$ be a relation from a set of variables X to a domain D. We denote $\sigma_F(R)$ as the subset of R where the formula F is true. Throughout the paper, we will assume that both the domain D and the variable set X are ordered. The order on D will be denoted as $<$ and the order on X as $\prec$ and we will often write $D = \{d_1, \dots, d_p\}$ with $d_1 < \cdots < d_p$ and $X = \{x_1, \dots, x_n\}$ with $x_1 \prec \cdots \prec x_n$. Given $d \in D$, we denote by $\mathsf{rank}(d)$ the number of elements of D that are smaller or equal to d. Both $<$ and $\prec$ induce a lexicographical order $\prec_{\mathsf{lex}}$ on D^X defined as $\tau \prec_{\mathsf{lex}} \sigma$ if there exists $x \in X$ such that for every $y \prec x$, $\tau(y) = \sigma(y)$ and $\tau(x) < \sigma(x)$. Given an integer $k \leqslant \#R$, we denote by $R[k]$ the k^{th} tuple in R for the $\prec_{\mathsf{lex}}$-order. The following observation will prove useful to design a DA algorithm:

▶ **Lemma 1 ($\star$).** *Let $\tau = R[k]$ and $x = \mathsf{min}(\mathsf{var}(R))$. Then $\tau(x) = \mathsf{min}\{d \mid \#\sigma_{x \leqslant d}(R) \geqslant k\}$. Moreover, $\tau = R'[k']$, where $R' = \sigma_{x=d}(R)$ and $k' = k - \#\sigma_{x<d}(R)$.*

Figure 1 gives an intuition of the result presented in Lemma 1 as a visual represention of the index transformation.

Queries. A *positive atom* (resp. *negative atom*) is an expression of $R(\mathbf{x})$ (resp. $\neg R(\mathbf{x})$) where R is a relation symbol and $\mathbf{x}$ a tuple of variables in X. A *signed join query* Q is a set of (negative or positive) atoms $Q = \{R_1(\mathbf{x_1}), \dots, R_p(\mathbf{x_p}), \neg S_{p+1}(\mathbf{x_{p+1}}), \dots, \neg S_m(\mathbf{x_m})\}$. In this paper, we consider *self-join free* queries, that is, we assume that two distinct atoms of a join query have distinct relation symbols. The set of variables of Q is denoted by $\mathsf{var}(Q)$, the set of positive (resp. negative) atoms of Q is denoted by $\mathsf{atoms}^+(Q)$ (resp. $\mathsf{atoms}^-(Q)$). A *positive (resp. negative) join query* is a signed join query without negative (resp. positive) atoms. The size $|Q|$ of Q is defined as $\sum_{i=1}^m |\mathbf{x_i}|$, where $|\mathbf{x}|$ denotes the number of variables in $\mathbf{x}$. A *database* $\mathbf{D}$ for Q is an ordered finite set D called the *domain* together with a set of

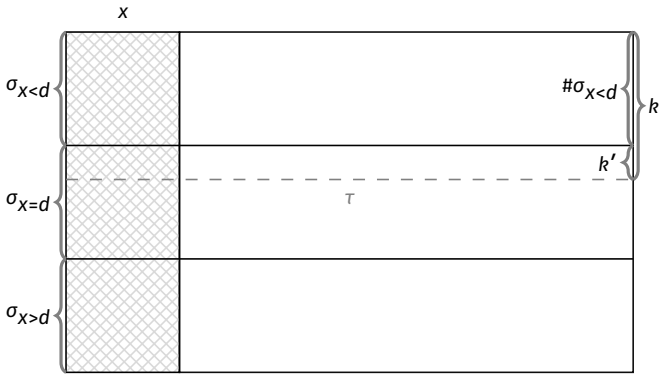


Figure 1 Representation of the link between k and k'.

relations $R_i^{\mathbf{D}} \subseteq D^{a_i}$, $S_j^{\mathbf{D}} \subseteq D^{a_j}$ such that $a_i = |\mathbf{x_i}|$. The *answers of* Q *over* $\mathbf{D}$ is the relation $[\![Q]\!]^{\mathbf{D}} \subseteq D^{\mathsf{var}(Q)}$ defined as the set of $\sigma \in D^{\mathsf{var}(Q)}$ such that for every $i \leqslant p$, $\sigma(\mathbf{x_i}) \in R_i^{\mathbf{D}}$ and for every $p < i \leqslant m$, $\sigma(\mathbf{x_i}) \notin S_i^{\mathbf{D}}$. The size $|\mathbf{D}|$ of the database $\mathbf{D}$ is defined to be the total number of tuples in it plus the size of its domain, that is, $|D| + \sum_{i=1}^{\ell} |R_i| + \sum_{i=\ell+1}^{m} |S_j|$. We follow the definition of [25] about the size of the database. Adding the size of the domain here is essential since we are dealing with negative atoms. Hence a query may have answers even when the database is empty, for example, $Q = \neg R(x)$ with $R^{\mathbf{D}} = \emptyset$ has $|D|$ answers.

A *signed conjunctive query* $Q(Y)$ is a join query Q together with $Y \subseteq \mathsf{var}(Q)$, called the *free variables of* Q and denoted by $\mathsf{free}(Q)$. The answers $[\![Q(Y)]\!]^{\mathbf{D}}$ of a conjunctive query Q over a database D are defined as $[\![Q]\!]^{\mathbf{D}}|_Y$, that is, the projection over Y of answers of Q.

Direct Access tasks. Given a query Q, a database instance $\mathbf{D}$ on an ordered domain D and a total order $\prec$ on the variables of Q, a *Direct Access (DA for short) task* [12] is the problem of returning, on input k, the k-th tuple $[\![Q]\!]^{\mathbf{D}}[k]$ for the order $\prec_{\mathsf{lex}}$ if $k < \#[\![Q]\!]^{\mathbf{D}}$ and fail otherwise. We are interested in answering DA tasks using the same setting as [12]: we allow a *precomputation* phase during which a data structure is constructed, followed by an *access* phase. Our goal is to obtain an algorithm for DA tasks with polynomial precomputation time and access time that is polylogarithmic time in the size of $\mathbf{D}$.

Hypergraphs and Signed Hypergraphs. A *hypergraph* $H = (V, E)$ is defined as a set of *vertices* V and *hyperedges* $E \subseteq 2^V$, that is, a hyperedge $e \in E$ is a subset of V. A *signed hypergraph* $H = (V, E_+, E_-)$ is defined as a set of *vertices* V, *positive edges* $E_+ \subseteq 2^V$ and *negative edges* $E_- \subseteq 2^V$. The *signed hypergraph* $H(Q) = (\mathsf{var}(Q), E_+, E_-)$ *of a signed conjunctive query* $Q(Y)$ is defined as the signed hypergraph whose vertex set is the variables of Q and such that $E_+ = \{\mathsf{var}(a) \mid a \text{ is a positive atom of } Q\}$ and $E_- = \{\mathsf{var}(a) \mid a \text{ is a negative atom of } Q\}$.

A *subhypergraph* H' of H, denoted by $H' \subseteq H$ is a hypergraph of the form (V, E') with $E' \subseteq E$. For $S \subseteq V$, we denote by $H \setminus S$ the hypergraph $(V \setminus S, E')$ where $E' = \{e \setminus S \mid e \in E\}$. Given $v \in V$, we denote by $E(v) = \{e \in E \mid v \in e\}$ the set of edges containing v, by $N_H(v) = \bigcup_{e \in E(v)} e$ the *neighbourhood of* v *in* H and by $N_H^*(v) = N_H(v) \setminus \{v\}$ the *open neighbourhood of* v. We will be interested in the following vertex removal operation on H: given a vertex v of H, we denote by $H/v = (V \setminus \{v\}, E/v)$ where E/v is defined as $\{e \setminus \{v\} \mid e \in E\} \setminus \{\emptyset\} \cup N_H^*(v)$, that is, H/v is obtained from H by removing v from every edge of H and by adding a new edge that contains the open neighbourhood of v.

Given $S \subseteq V$ and $E \subseteq 2^V$, a *covering of S with E* is a subset $F \subseteq E$ such that $S \subseteq \bigcup_{e \in F} e$. The *cover number $cn(S, E)$ of S wrt E* is defined as the minimal size of a covering of S with E, that is, $cn(S, E) = \min\{|F| \mid F \text{ is a covering of } S \text{ with } E\}$. A *fractional covering of S with E* is a function $c : E \to \mathbb{R}_+$ such that for every $s \in S$, $\sum_{e \in E(s)} c(e) \geqslant 1$. Observe that a covering is a fractional covering where c has values in $\{0, 1\}$. The *fractional cover number $fcn(S, E)$ of S wrt E* is defined as the minimal size of a fractional covering of S with E, that is, $fcn(S, E) = \min\{\sum_{e \in E} c(e) \mid c \text{ is a fractional covering of } S \text{ with } E\}$. Fractional covers are particularly interesting because of the following theorem by Grohe and Marx:

▶ **Theorem 2** ([19]). *Let Q be a join query and λ be the fractional cover number of $\mathsf{var}(Q)$. Then for every database $\mathbf{D}$, $[\![Q]\!]^{\mathbf{D}}$ has size at most $|\mathbf{D}|^{\lambda}$.*

3 Ordered relational circuits

In this section, we introduce a data structure that can be used to succinctly represent relations. This data structure is an example of a factorised representation, such as d-representations [28], but does not need to be structured along a tree, which will allow us to handle more queries, and especially queries with negative atoms – for example β-acyclic signed conjunctive queries, a class of queries that do not have polynomial size d-representations [9, Theorem 9].

3.1 Definitions

Relational circuits. A $\{\bowtie, \mathsf{dec}\}$-circuit C on variables $X = \{x_1, \ldots, x_n\}$ and domain D is a multi-directed acyclic graph, that is, there may be more than one edge between two nodes u and v, with one special gate $\mathsf{out}(C)$ called the *output* of C. The circuit is labelled as follows:

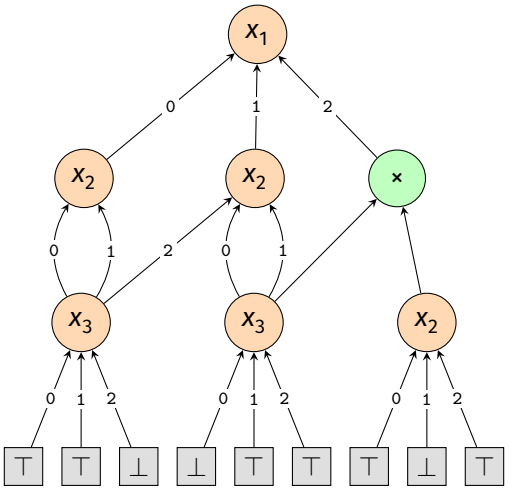
- every gate of C with no ingoing edge, called an *input of C*, is labelled by either $\bot$ or $\top$;
- a gate v labelled by a variable $x \in X$ is called a *decision gate*. We denote x by $\mathsf{decvar}(v)$. Each ingoing edge e of v is labelled by a value $d \in D$ and for each $d \in D$, there is at most one ingoing edge of v labelled by d. This implies that a decision gate has at most $|D|$ ingoing edges; and
- every other gate is labelled by $\bowtie$.

The set of all the decision gates in a circuit C is denoted by $\mathsf{decision}(C)$. Given a gate v of C, we denote by C_v the *subcircuit of C rooted in v* to be the circuit whose gates are the gates reachable from v by following a directed path in C. We define the *variable set of v*, denoted by $\mathsf{var}(v) \subseteq X$, to be the set of variables x labelling a decision gate in C_v. The *size* $|C|$ of a $\{\bowtie, \mathsf{dec}\}$-circuit is defined to be the number of edges of its underlying DAG. In the example circuit of Figure 2, the decision gates are represented as containing the variable that labels them. Considering v to be the leftmost x_2 decision gate, we have $\mathsf{var}(v) = \{x_2, x_3\}$.

We define the *relation* $\mathsf{rel}(v) \subseteq D^{\mathsf{var}(v)}$ *computed at gate v* inductively as follows: if v is an input labelled by $\bot$, then $\mathsf{rel}(v) = \emptyset$. If v is an input labelled by $\top$, then $\mathsf{rel}(v) = D^{\emptyset}$, that is, $\mathsf{rel}(v)$ is the relation containing only the empty tuple. Otherwise, let $v_1, \ldots, v_k$ be the inputs of v. If v is a $\bowtie$-gate, then $\mathsf{rel}(v)$ is defined to be $\mathsf{rel}(v_1) \bowtie \ldots \bowtie \mathsf{rel}(v_k)$. If v is a decision gate labelled by a variable x, $\mathsf{rel}(v) = \big([x \leftarrow d_1] \bowtie \mathsf{rel}(v_1) \times D^{\Delta(v,v_1)}\big) \cup \cdots \cup \big([x \leftarrow d_k] \bowtie \mathsf{rel}(v_k) \times D^{\Delta(v,v_k)}\big)$ where e_i is the incoming edge (v_i, v) labelled by d_i and $\Delta(v, v_i) = \mathsf{var}(v) \setminus (\{x\} \cup \mathsf{var}(v_i))$. It is readily verified that $\mathsf{rel}(v)$ is a relation on domain D and variables $\mathsf{var}(v)$. The *relation computed by C over a set of variables X* (assuming $\mathsf{var}(C) \subseteq X$), denoted by $\mathsf{rel}_X(C)$, is defined to be $\mathsf{rel}(\mathsf{out}(C)) \times D^{X \setminus \mathsf{var}(\mathsf{out}(C))}$. In Figure 2, if v is the leftmost x_3 decision gate, then $\mathsf{rel}(v)$ is the relation where x_3 is set to 0 or 1.

Deciding whether the relation computed by a $\{\bowtie, \mathsf{dec}\}$-circuit is non-empty is NP-complete by a straightforward reduction to testing non-emptiness of conjunctive queries [14]. Such circuits are hence of little use to get tractability results. We are therefore more interested in the following restriction of $\{\bowtie, \mathsf{dec}\}$-circuits where testing non-emptiness and other related tasks are tractable: a $\{\times, \mathsf{dec}\}$-circuit C is a $\{\bowtie, \mathsf{dec}\}$-circuit such that: (i) for every $\bowtie$-gate v of C with inputs $v_1, \ldots, v_k$ and $i < j \leqslant k$, it holds that $\mathsf{var}(v_i) \cap \mathsf{var}(v_j) = \emptyset$, (ii) for every decision gate v of C labelled by x with inputs $v_1, \ldots, v_k$ and $i \leqslant k$, it holds that $x \notin \mathsf{var}(v_i)$. An example of such a circuit is presented in Figure 2.

Let X be a set of variables ordered by $\prec$. A $\{\times, \mathsf{dec}\}$-circuit C on domain D and variables X is a *$\prec$-ordered $\{\times, \mathsf{dec}\}$-circuit* if for every decision gate v of C labelled with $x \in X$, it holds that for every $y \in \mathsf{var}(v) \setminus \{x\}$, $x \prec y$. We simply say that a circuit C is an ordered $\{\times, \mathsf{dec}\}$-circuit if there exists some order $\prec$ on X such that C is a $\prec$-ordered $\{\times, \mathsf{dec}\}$-circuit. Observe that if v is a decision-gate in an ordered $\{\times, \mathsf{dec}\}$-circuit, then $\mathsf{rel}(v) = \left([x \leftarrow d_1] \times \mathsf{rel}(v_1) \times D^{\Delta(v,v_1)}\right) \uplus \cdots \uplus \left([x \leftarrow d_k] \times \mathsf{rel}(v_k) \times D^{\Delta(v,v_k)}\right)$ since $x \notin \mathsf{var}(v_i)$ by definition and that these unions are disjoint because of the value assigned to x. The circuit presented in Figure 2 is an ordered $\{\times, \mathsf{dec}\}$-circuit for the order induced by the variables x_1, x_2, x_3.



(a) ordered $\{\times, \mathsf{dec}\}$-circuit.

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	1
1	0	2
1	1	1
1	1	2
1	2	0
1	2	1
2	0	1
2	0	2
2	2	1
2	2	2

(b) relation computed by the circuit.

Figure 2 Example of a small ordered $\{\times, \mathsf{dec}\}$-circuit computing a simple relation.

Frontiers. Let $X = \{x_1, \ldots, x_n\}$ with $x_1 \prec \cdots \prec x_n$. A *prefix assignment* of size p is an assignment of variables $\tau \in D^{\{x_1, \ldots, x_p\}}$ with $p \leqslant n$. When answering DA tasks, we will need to be able to find a representation of the tuples in $\mathsf{rel}(C)$ that agree with τ. To do that, we introduce the notion of frontier. Given τ a prefix assignment, the *frontier f_τ of τ* in C is defined algorithmically as follows:

1. Instantiate a set F with $\mathsf{out}(C)$, the root of the circuit.
2. As long as F is not stable, do:
   - if $v \in F$ is a $\times$-gate, $F \coloneqq (F \setminus \{v\}) \cup \bigcup_{w \in \mathsf{input}(v)} w$,
   - if $v \in F$ is a decision gate and the variable x labelling v is assigned in the prefix $(x \in \{x_1, \ldots, x_p\})$, $F \coloneqq (F \setminus \{v\}) \cup \{v'\}\}$ where v' is the node connected to v by the edge (v', v) labelled by $\tau(x)$.
3. If F contains a $\bot$-gate, then $f_\tau = \{\bot\}$, otherwise $f_\tau = F$.

Frontiers are particularly useful because they can be efficiently computed and the relation they represent is essentially the tuples of the relation represented by C that agree with τ. We denote by $\mathsf{var}(f_\tau) = \bigcup_{v \in f_\tau} \mathsf{var}(v)$ the set of variables appearing below a gate in the frontier and by $\mathsf{rel}(f_\tau) = \bigtimes_{v \in f_\tau} \mathsf{rel}(v)$. Given a prefix assignment τ of variables $\{x_1, \ldots, x_p\}$ and a relation R, we slightly abuse notation by denoting $\sigma_\tau(R)$ the relation $\sigma_{x_1=\tau(x_1) \wedge \cdots \wedge x_p=\tau(x_p)}(R)$. We can prove by induction on p that:

▶ **Lemma 3** (⋆). *Let τ be a prefix assignment on variables $x_1, \ldots, x_p$. Then f_τ can be computed in $\mathcal{O}(|X|)$ and $\sigma_\tau(\mathsf{rel}_X(C)) = \{\tau\} \times \mathsf{rel}(f_\tau) \times D^{\{x_{p+1}, \ldots, x_n\} \setminus \mathsf{var}(f_\tau)}$.*

3.2 Direct Access for ordered $\{\times, \mathsf{dec}\}$-circuits

The main result of this section is an algorithm that allows for efficient solving of DA tasks for an ordered $\{\times, \mathsf{dec}\}$-circuit on domain D and variables X. More precisely, we prove:

▶ **Theorem 4.** *Let $(X, \prec)$ be a finite ordered set and C a $\prec$-ordered $\{\times, \mathsf{dec}\}$-circuit on variables X and domain D. We can answer direct access tasks on $\mathsf{rel}_X(C)$ ordered by $\prec_{\mathsf{lex}}$ with precomputation time $\mathcal{O}(|C| \cdot \mathsf{poly}(|X|) \cdot \mathsf{polylog}|D|)$ and access time $\mathcal{O}(\mathsf{poly}(|X|) \cdot \mathsf{polylog}|D|)$.*

Precomputation. In this section, we assume that C is a $\prec$-ordered $\{\times, \mathsf{dec}\}$-circuit on variables X. The *count label of* C, denoted by nrel_C, is the mapping from $\mathsf{decision}(C) \times D$ to $\mathbb{N}$ such that $\mathsf{nrel}_C(v, d) = \#\sigma_{x \leqslant d}(\mathsf{rel}(v))$, that is, $\mathsf{nrel}_C(v, d)$ is the number of tuples from $\mathsf{rel}(v)$ that assign a value on x smaller or equal than d. The precomputation step aims to compute nrel_C so that we can access $\mathsf{nrel}_C(v, d)$ quickly.

Our algorithm performs a bottom-up computation of the number of satisfying tuples in $\mathsf{rel}(v)$ for every gate v of C. If $\mathsf{rel}(v)$ is a decision-gate on variable x, we have by definition:

$$|\mathsf{rel}(v)| = \sum_{w \in \mathsf{input}(v)} |\mathsf{rel}(w)| \times |D|^{|\Delta(v,w)|} \text{ where } \Delta(v, w) = \mathsf{var}(v) \setminus (\{x\} \cup \mathsf{var}(w)). \tag{1}$$

Similarly, $\mathsf{nrel}_C(v, d)$ can be computed by restricting the previous relation on the inputs of v that sets x to a value $d' \leqslant d$, that is:

$$\mathsf{nrel}_C(v, d) = \sum_{w \in \mathsf{input}(v), d_w \leqslant d} |\mathsf{rel}(w)| \times |D|^{|\Delta(v,w)|} \text{ where } d_w \text{ is the label of edge } (v, w). \tag{2}$$

Finally observe that if v is a $\times$-gate, we clearly have $|\mathsf{rel}(v)| = \prod_{w \in \mathsf{input}(v)} |\mathsf{rel}(w)|$ and for every gate v, $\mathsf{var}(v) = \bigcup_{w \in \mathsf{input}(v)} \mathsf{var}(w)$.

Hence, using a dynamic programming algorithm that follows the structure of the circuit in a bottom-up way, we can compute tables T_{rel}, T_{var} and T_{nrel_C} such that $T_{\mathsf{rel}}[v] = |\mathsf{rel}(v)|$, $T_{\mathsf{var}}[v] = \mathsf{var}(v)$ and $T_{\mathsf{nrel}_C}[v, d] = \mathsf{nrel}_C(v, d)$ for every gate v and value $d \in D$, allowing us to prove the following (a full description of the dynamic programming algorithm can be found in the full version):

▶ **Lemma 5** (⋆). *Given a $\prec$-ordered $\{\times, \mathsf{dec}\}$-circuit C, we can compute a data structure in time $\mathcal{O}(|C| \cdot \mathsf{poly}|X|\mathsf{polylog}|D|)$ that allows us to access $\mathsf{var}(v)$, $|\mathsf{rel}(v)|$ for every gate v of C in time $\mathcal{O}(1)$ and $\mathsf{nrel}_C(v, d)$ for every decision gate v and $d \in D$ in time $\mathcal{O}(\mathsf{polylog}(|D|))$.*

An example of what the nrel_C values might look like in practice can be seen at the left of the decision gates in Figure 3.

Direct access. We now show how the precomputation from Lemma 5 allows us to get direct access for ordered $\{\times, \mathsf{dec}\}$-circuits. We first show how one can solve a DA task for any relation as long as we have access to very simple counting oracles. We then show that one can quickly simulate these oracle calls in ordered $\{\times, \mathsf{dec}\}$-circuits using precomputed values.

We start by illustrating our algorithm on an example.

▶ **Example 6.** Consider Figure 3, which represents two different direct access tasks on a circuit where nrel_C has been precomputed and which is represented as lists beside each node.

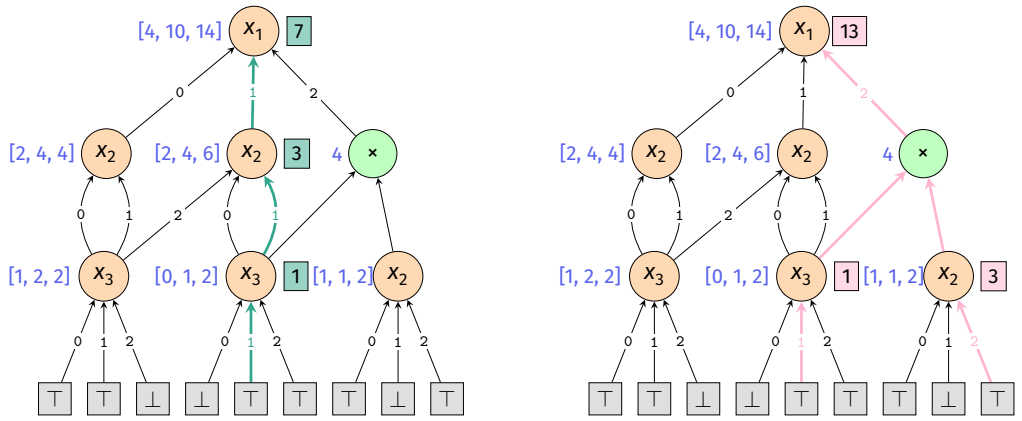
We start by explaining how we solve direct access for $k = 7$, depicted in Figure 3a. In this paragraph, let τ_7 be the 7-th solution of the circuit, that is, $\{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 1\}$ (see Figure 2). Our algorithm iteratively finds the values of τ_7 on x_1, x_2 and x_3. It starts by finding the value $\tau_7(x_1)$ from the decision node at the root of the circuit. Using nrel_C, one can see that the smallest value one can assign to x_1 to have at least 7 solutions is 1. By Lemma 1, we know that $\tau_7(x_1) = 1$. The algorithm then follows the edge labelled by 1 to find a subcircuit computing every solution of the circuit when x_1 is fixed to 1. We can now repeat the method to find $\tau_7(x_2)$. However, one has to be careful about the index. By setting x_1 to 1, we have discarded every solution of the circuit where $x_1 < 1$, that is, every solution where $x_1 \leqslant 0$. From nrel_C, we know that there are 4 such solutions. Hence, when repeating the method, we now look for the third $(7 - 4 = 3)$ solution of the next gate. The same reasoning yields that $\tau_7(x_2) = 1$ and that we discard 2 solutions. Applying this method once more yields the desired value for x_3.

The algorithm is a bit more complex when encountering $\times$-gates. We illustrate it in Figure 3b, where we solve direct access for $k = 13$. Again, we denote by τ_{13} the 13-th solution of the circuit. We find that $\tau_{13}(x_1) = 2$ exactly as before. We know that setting x_1 to 2 discards 10 solutions. Hence, we need to find the 3-rd solution of the circuit rooted in the $\times$-gate. To do that, we first follow every edge going in the gate to find only decision-gates. By 3, we know that the solution of the $\times$-gate are the Cartesian product of the solutions of both decision-gates. Now to find the value $\tau_{13}(x_2)$, we do the same reasoning as before but one has to be careful: nrel_C only contains the number of solutions of the subcircuit but each one of these solutions can be extended to a full solution of the circuit by completing it with the value of x_3. We now from nrel_C on the decision-gate labelled by x_3 that there are 2 such solutions. Hence, each solution of the decision-gate labelled by x_2 can be extended into 2 full solutions. Hence, we know that there are 2 solutions where $x_2 = 0$ and 4 solutions where $x_2 = 2$. Hence, similarly as before, we can deduce that $\tau_{13}(x_2) = 2$. We find the value of x_3 similarly as before.

▶ **Lemma 7** ($\star$). *Assume that we are given a relation $R \subseteq D^X$ with $X = \{x_1, \ldots, x_n\}$ and an oracle such that for every prefix assignment τ of size p it returns $\#\sigma_{\tau \wedge x_{p+1} \leqslant d}(R)$. Then, for any k, we can compute $R[k]$ using $\mathcal{O}(n \cdot \mathsf{polylog}|D|)$ oracle calls, where $n = |X|$.*

Proof sketch. Let $\tau = R[k]$. We apply Lemma 1 iteratively to find $\tau(x_i)$ for each $i = 1, \ldots, n$. By Lemma 1, we know that $\tau(x_1) = d$ where d is the smallest value in D such that $\#\sigma_{x_1 \leqslant d}(R) \geqslant k$. Now one can easily find d using a binary search that makes $\mathcal{O}(\mathsf{log}|D|)$ oracle calls. Now, finding $\tau(x_2)$ boils down to find $R'[k']$ where the $k' = k - \#\sigma_{x_1 < d}(R)$ and $R' = \sigma_{x_1 = d}(R)$ which can be done similarly using oracle calls for R', which could be simulated by oracle calls to R, and so on for each value $\tau(x_i)$. ◀

Now, it remains to show that oracle calls as in Lemma 7 can be efficiently solved on ordered $\{\times, \mathsf{dec}\}$-circuits after preprocessing.



(a) $k = 7$. **(b)** $k = 13$.

Figure 3 Examples of paths followed in the circuit for different direct access tasks. Precomputed values of nrel_C for decision gates are represented as lists at the left of each gate. The current index of the tuple we are searching for is indicated next to the reached nodes.

▶ **Lemma 8** (⋆). *For a given prefix assignment τ of size p and $d \in D$, assuming that $|\mathsf{rel}(v)|$, $\mathsf{nrel}_C(v, d)$ and $\mathsf{var}(v)$ have been precomputed for every gate v in C, $\#\sigma_{\tau \wedge x_{p+1} \leqslant d}(\mathsf{rel}(C))$ can be computed in time $\mathcal{O}(\mathsf{poly}(n)\mathsf{polylog}|D|)$, where $n = |X|$.*

Proof sketch. To ease notation, let $x = x_{p+1}$. We first compute f_τ in $\mathcal{O}(n)$ with Lemma 3. Now, $\sigma_\tau(\mathsf{rel}(C) = D^\Delta \times \mathsf{rel}(f_\tau) \times \{\tau\}$ for $\Delta = \{x_{p+1}, \ldots, x_n\} \setminus \mathsf{var}(f_\tau)$. Now, either $x \in \Delta$ and we have $\sigma_{\tau \wedge x \leqslant d}(\mathsf{rel}(C)) = D^{\Delta \setminus \{x\}} \times \sigma_{x \leqslant d}(D^{\{x\}}) \times \mathsf{rel}(f_\tau) \times \{\tau\}$ or $x_{p+1} \notin \Delta$ and we can show that there exists a decision gate v in f_τ such that $\mathsf{decvar}(v) = x$. In this case, $\sigma_{\tau \wedge x \leqslant d}(\mathsf{rel}(C)) = D^\Delta \times \sigma_{x \leqslant d}(\mathsf{rel}(v)) \times \mathsf{rel}(f_\tau \setminus \{v\}) \times \{\tau\}$. In any case, we can compute each part of the Cartesian product from precomputed values: $|\mathsf{rel}(f_\tau)| = \prod_{w \in f_\tau} |\mathsf{rel}(w)|$, $|D^\Delta| = |D|^{|\Delta|}$, $\#\sigma_{x \leqslant d}(D^{\{x\}}) = \mathsf{rank}(d)$ and $\#\sigma_{x \leqslant d}(\mathsf{rel}(v)) = \mathsf{nrel}_C(v, d)$, hence $\#\sigma_{\tau \wedge x_{p+1} \leqslant d}(\mathsf{rel}(C))$ is computed with at most n multiplications of integers of size at most $n \cdot \mathsf{polylog}(|D|)$. ◀

Theorem 4 now follows from Lemmas 5, 7, and 8. Indeed, after preprocessing the circuit using Lemma 8, one can use the procedure described in Lemma 7 to solve direct access task using oracle calls, that can be answered efficiently as shown in Lemma 8.

4 From join queries to ordered $\{\times, \mathsf{dec}\}$-circuits

In this section, we present a simple top-down algorithm such that on input Q, $\prec$ and $\mathbf{D}$, it returns a $\succ$-ordered $\{\times, \mathsf{dec}\}$-circuit C such that $\mathsf{rel}(C) = Q(\mathbf{D})$, where Q is a join query, $\prec$ an order on its variables and $\mathbf{D}$ a database. This algorithm is an adaptation of exhaustive DPLL [31], an algorithm that has been originally devised to solve the #SAT problem, but Huang and Darwiche [20] have shown that the trace of this algorithm implicitly builds a Boolean circuit, corresponding to the $\{\times, \mathsf{dec}\}$-circuits on domain $\{0, 1\}$. We show how to adapt it in the framework of signed join queries. The algorithm itself is presented in Section 4.1. We study the complexity of this algorithm in Section 4.3 depending on the structure of Q and $\prec$, using hypergraph structural parameters introduced in Section 4.2.

4.1 Exhaustive DPLL for signed join queries

The main idea of DPLL for signed join queries is the following: given an order $\prec$ on the variables of a join query Q and a database $\mathbf{D}$, we construct a $\succ$-ordered $\{\times, \mathsf{dec}\}$-circuit (where $x \succ y$ iff $y \prec x$)[1] computing $[\![Q]\!]^{\mathbf{D}}$ by successively testing the variables of Q with decision gates, from the highest to the lowest wrt $\prec$. At its simplest form, the algorithm picks the highest variable x of Q wrt $\prec$, creates a new decision gate v on x and then, for every value $d \in D$, sets x to d and recursively computes a gate v_d computing the subset of $[\![Q]\!]^{\mathbf{D}}$ where $x = d$. We then add v_d as an input of v and proceed with the next value $d' \in D$. This approach is however not enough to get interesting tractability results. We hence add the following optimisations. First, we keep a cache of already computed queries so that if we recursively call the algorithm twice on the same input, we can directly return the previously constructed gate. Moreover, if we detect that the answers of Q are the Cartesian product of two or more subqueries $Q_1, \ldots, Q_k$, then we create a new $\times$-gate v, recursively call the algorithm on each component Q_i to construct a gate w_i and plug each w_i to v. Detecting such cases is mainly done syntactically, by checking whether the query can be partitioned into subqueries having disjoint variables. However, this approach would fail to give good complexity bounds in the presence of negative atoms. To achieve the best complexity, we also remove from Q every negative atom as soon as it is satisfied by the current partial assignment. This allows us to discover more cases where the query has connected components. The theoretical performance of the previously described algorithm may however vary if one is not careful in the way the recursive calls are actually made. We hence give a more formal presentation of the algorithm, whose pseudocode is presented in Algorithm 1.

Algorithm 1 An algorithm to compute a $\succ$-ordered $\{\times, \mathsf{dec}\}$-circuit representing $[\![Q]\!]^{\mathbf{D}}$.

procedure DPLL($Q, \tau, \mathbf{D}, \prec$)
  if (Q, τ) is in cache **then return** cache(Q, τ)
  if Q is inconsistent with τ **then return** $\bot$-gate
  if τ assigns every variable in Q **then return** $\top$-gate
  $x \leftarrow \max_{\prec} \mathsf{var}(Q)$
  for $d \in D$ **do**
    $\tau' \leftarrow \tau \times [x \leftarrow d]$
    if Q is inconsistent with τ' **then** $v_d \leftarrow \bot$-gate
    else
      Let $Q_1, \ldots, Q_k$ be the τ'-connected components of $Q \Downarrow \tau'$
      for $i = 1$ *to* k **do** $w_i \leftarrow$ DPLL$(Q_i, \tau_i, \mathbf{D}, \prec)$ where $\tau_i = \tau'|_{\mathsf{var}(Q_i)}$
      $v_d \leftarrow$ **new** $\times$-gate with inputs $w_1, \ldots, w_k$
    end if
  end for
  $v \leftarrow$ **new** dec-gate connected to v_d by a d-labelled edge for every $d \in D$
  cache$(Q, \tau) \leftarrow v$
  return v
end procedure

[1] While one could easily change the algorithm so that it produces a $\prec$-ordered $\{\times, \mathsf{dec}\}$-circuit instead, the structural parameters we will be considering for the tractability of DPLL in Section 4.2 are more naturally defined on $\prec$. We choose to present DPLL this way to ease the proofs later.

A few notations are used in Algorithm 1. Given a database $\mathbf{D}$ on domain D and a tuple $\tau \in D^Y$, we denote by $[\![Q]\!]^{\mathbf{D}}_\tau$ the set of tuples $\sigma \in D^{\mathsf{var}(Q)\setminus Y}$ that are answers of Q when extended with τ. More formally, $\sigma \in [\![Q]\!]^{\mathbf{D}}_\tau$ if and only if $(\sigma \times \tau)|_{\mathsf{var}(Q)} \in [\![Q]\!]^{\mathbf{D}}$. Given an atom $R(\mathbf{x})$, a database $\mathbf{D}$ and a tuple $\tau \in D^Y$, we say that $R(\mathbf{x})$ *is inconsistent with* τ *wrt* $\mathbf{D}$ (or simply inconsistent with τ when $\mathbf{D}$ is clear from context) if there is no $\sigma \in R^{\mathbf{D}}$ such that $\tau \simeq \sigma$. Observe that if Q contains a positive atom $R(\mathbf{x})$ that is inconsistent with τ then $[\![Q]\!]^{\mathbf{D}}_\tau = \emptyset$. Similarly, if Q contains a negative atom $\neg R(\mathbf{x})$ such that τ assigns every variable of $\mathbf{x}$ and $\tau(\mathbf{x}) \in R$, then $[\![Q]\!]^{\mathbf{D}}_\tau = \emptyset$. If one of these cases arises, we say that Q is *inconsistent with* τ. Now observe that if $\neg R(\mathbf{x})$ is a negative atom of Q such that $R(\mathbf{x})$ is inconsistent with τ, then $[\![Q]\!]^{\mathbf{D}}_\tau = [\![Q']\!]^{\mathbf{D}}_\tau \times D^W$ where $Q' = Q \setminus \{\neg R(\mathbf{x})\}$ and $W = \mathsf{var}(Q) \setminus \mathsf{var}(Q')$ (some variables of Q may only appear in the atom $\neg R(\mathbf{x})$). This motivates the following definition: the *simplification of* Q *wrt to* τ *and* $\mathbf{D}$, denoted by $Q \Downarrow \langle \tau, \mathbf{D}\rangle$ or simply by $Q \Downarrow \tau$ when $\mathbf{D}$ is clear from context, is defined to be the subquery of Q obtained by removing from Q every negative atom $\neg R(\mathbf{x})$ of Q such that $R(\mathbf{x})$ is inconsistent with τ. From what precedes, we clearly have $[\![Q]\!]^{\mathbf{D}}_\tau = [\![Q']\!]^{\mathbf{D}}_\tau \times D^W$ where $Q' = Q \Downarrow \langle \tau, \mathbf{D}\rangle$ and $W = \mathsf{var}(Q) \setminus \mathsf{var}(Q')$.

For a tuple $\tau \in D^Y$ assigning a subset Y of variables of Q, the *τ-intersection graph $\mathcal{I}^Q_\tau$ of Q* is the graph whose vertices are the atoms of Q having at least one variable not in Y and there is an edge between two atoms a, b of Q if a and b share a variable that is not in Y. Observe that $\mathcal{I}^Q_\tau$ does not depend on the values of τ but only on the variables it sets. Hence it can be computed in polynomial time in the size of Q only. A connected component C of $\mathcal{I}^Q_\tau$ naturally induces a subquery Q_C of Q and is called a *τ-connected component*. Q is partitioned into its τ-connected components and the atoms whose variables are completely set by τ. More precisely, $Q = \bigcup_{C\in\mathcal{CC}} Q_C \cup Q'$ where $\mathcal{CC}$ are the connected component of $\mathcal{I}^Q_\tau$ and Q' contains every atom a of Q on variables $\mathbf{x}$ such that $\mathbf{x}$ only has variables in Y. Observe that if τ is an answer of Q', then $[\![Q]\!]^{\mathbf{D}}_\tau = \bigtimes_{C\in\mathcal{CC}} [\![Q_C]\!]^{\mathbf{D}}_{\tau_C}$ where $\tau_C = \tau|_{\mathsf{var}(Q_C)}$ since if C_1 and C_2 are two distinct τ-connected components of $\mathcal{I}^Q_\tau$, then $\mathsf{var}(Q_{C_1}) \cap \mathsf{var}(Q_{C_2}) \subseteq Y$.

▶ **Example 9.** We illustrate the previous definitions on the signed join query $Q(x_1, \ldots, x_5)$ defined as $\neg R(x_1, \ldots, x_5), S(x_1, x_2, x_3), T(x_1, x_4, x_5)$ and database $\mathbf{D}$ on domain $\{0, 1\}$ with $R^{\mathbf{D}} = \{(1, 1, 1, 1, 1)\}$. Let $\tau = [x_1 \leftarrow 0]$. The τ-intersection graph of Q is a path where $\neg R(x_1, \ldots, x_5)$ is connected to $S(x_1, x_2, x_3)$ and $T(x_1, x_4, x_5)$. There is no edge between $S(x_1, x_2, x_3)$ and $T(x_1, x_4, x_5)$ since x_1 is their only common variable and it is assigned by τ. Hence, Q has one τ-connected component containing every atom of Q. Now, $Q \Downarrow \tau = S(x_1, x_2, x_3), T(x_1, x_4, x_5)$ since $R(0, x_2, \ldots, x_5)$ is inconsistent over $\mathbf{D}$ and the τ-intersection graph of $Q \Downarrow \tau$ consists in two isolated vertices $S(x_1, x_2, x_3)$ and $T(x_1, x_4, x_5)$. Hence $Q \Downarrow \tau$ has two τ-connected components. This example also illustrates the role of simplification for discovering Cartesian products.

The correctness can be proven by induction: a recursive call $\mathsf{DPLL}(Q, \tau, \mathbf{D}, \prec)$ returns a gate computing $[\![Q]\!]^{\mathbf{D}}_\tau$. This is formalised in the following theorem. We analyse the complexity of DPLL in Section 4.3.

▶ **Theorem 10** ($\star$)**.** *Let Q be a signed join query, $\mathbf{D}$ a database and $\prec$ an order on $\mathsf{var}(Q)$, then $\mathsf{DPLL}(Q, \langle\rangle, \mathbf{D}, \prec)$ constructs a $\succ$-ordered $\{\times, \mathsf{dec}\}$-circuit C and returns a gate v of C such that $\mathsf{rel}(v) = [\![Q]\!]^{\mathbf{D}}$.*

4.2 Hyperorder width

In this section, we introduce the notions of width based on elimination orders rather than tree decompositions that are relevant to pinpoint the complexity of the DPLL procedure.

Order based widths ($\mathsf{how}(\cdot), \mathsf{fhow}(\cdot)$). A hypergraph $H = (V, E)$ and an order $\prec$ such that $V = \{v_1, \dots, v_n\}$ with $v_1 \prec \cdots \prec v_n$ induces a series of hypergraphs defined as $H_1^{\prec}, \dots, H_{n+1}^{\prec}$ as $H_1^{\prec} = H$ and $H_{i+1}^{\prec} = H_i^{\prec}/v_i$. The *hyperorder width* $\mathsf{how}(H, \prec)$ *of* $\prec$ *wrt* H is defined as $\max_{i \leqslant n} cn(N_{H_i^{\prec}}(v_i), E)$. The *hyperorder width* $\mathsf{how}(H)$ *of* H is defined as the best possible width using any elimination order, that is, $\mathsf{how}(H) = \min_{\prec} \mathsf{how}(H, \prec)$. We similarly define the *fractional hyperorder width* $\mathsf{fhow}(H, \prec)$ *of* $\prec$ *wrt* H as $\max_{i \leqslant n} fcn(N_{H_i^{\prec}}(v_i), E)$ and the *fractional hyperorder width* $\mathsf{fhow}(H)$ *of* H as $\mathsf{fhow}(H) = \min_{\prec} \mathsf{fhow}(H, \prec)$.

It has already been observed many times ([23, Appendix C] or [16, 17, 24]) that $\mathsf{how}(H)$ and $\mathsf{fhow}(H)$ are respectively equal to the generalised hypertree width and the fractional hypertree width of H and that there is a natural correspondence between a tree decomposition and an elimination order having the same width. However, to be able to express our tractability results as function of the order, it is more practical to define the width of orders instead of hypertree decompositions. In [7, Definition 9], $\mathsf{fhow}(H, \prec)$ is called the incompatibility number, though it is not formally defined on hypergraphs but directly on conjunctive queries. The case $k = 1$, which corresponds to the α-acyclicity of the underlying hypergraph, has also been previously called an order without disruptive trio [12]. However, these notions are specifically used for the problem of direct access in conjunctive queries while the characterisation of hypergraph measures in terms of elimination orders of hypergraphs predates by several years this terminology (see [5] for a survey). We then choose a terminology closer to the usual terminology for hypergraph decompositions.

Hereditary order based widths (β-$\mathsf{how}(\cdot), \beta$-$\mathsf{fhow}(\cdot)$). Hypertree width is not hereditary. That is, the (fractional) hypertree width of a subhypergraph can be much bigger than the (fractional) hypertree width of the hypergraph itself. It makes it not well suited to discover tractable classes for signed join queries. Indeed, if a query Q contains a negative atom $\neg R(\mathbf{x})$ and if $R^{\mathbf{D}}$ is empty in the database $\mathbf{D}$, then $[\![Q]\!]^{\mathbf{D}}$ is equal to $[\![Q']\!]^{\mathbf{D}}$, where $Q' = Q \setminus \{\neg R(\mathbf{x})\}$. Hence if some aggregation problem for a fixed self-join free query Q on an input database $\mathbf{D}$ can be solved in $O(\mathsf{poly}(|\mathbf{D}|))$ for any database $\mathbf{D}$, it has to be tractable for every Q' obtained by removing a subset of the negative atoms from Q. This motivates the following definitions: for a hypergraph $H = (V, E)$ and an order $\prec$ on V, the β-*hyperorder width* β-$\mathsf{how}(H, \prec)$ *of* $\prec$ *wrt to* H is defined as $\max_{H' \subseteq H} \mathsf{how}(H', \prec)$. The β-*hyperorder width* β-$\mathsf{how}(H)$ *of* H is defined as the width of the best possible elimination order, that is, β-$\mathsf{how}(H) = \min_{\prec} \beta$-$\mathsf{how}(H, \prec)$. We define similarly the β-fractional hyperorder width of an order $\prec$ and of an hypergraph – β-$\mathsf{fhow}(H, \prec)$ and β-$\mathsf{fhow}(H)$ – by replacing $\mathsf{how}(\cdot)$ by $\mathsf{fhow}(\cdot)$ in the definitions.

Comparison with existing measures. The fact that fractional hypertree width is not hereditary has traditionally been worked around by taking the largest width over every subhypergraph. In other words, the β-*fractional hypertree width* β-$\mathsf{fhtw}(H)$ of H is defined as β-$\mathsf{fhtw}(H) = \max_{H' \subseteq H} \mathsf{fhtw}(H')$. The β-hypertree width β-$\mathsf{htw}(H)$ is defined similarly. If one plugs the ordered characterisation of $\mathsf{fhtw}(H')$ in this definition, one can observe that β-$\mathsf{fhtw}(H) = \max_{H' \subseteq H} \min_{\prec} \mathsf{fhow}(H', \prec)$. Hence, the difference between β-$\mathsf{fhtw}(H)$ and β-$\mathsf{fhow}(H)$ boils down to inverting the min and the max in the definition. It directly gives that β-$\mathsf{fhtw}(H) \leqslant \beta$-$\mathsf{fhow}(H)$ and β-$\mathsf{htw}(H) \leqslant \beta$-$\mathsf{how}(H)$ for every H. The main advantage of the β-fractional hyperorder width is that it comes with a natural notion of decomposition – the best elimination order $\prec$ – that can be used algorithmically. This is not given by the definition of β-$\mathsf{fhtw}(\cdot)$ and has yet to be found.

The only exception is the case where β-$\mathsf{fhtw}(H) = 1$, known as β-acyclicity, where an order-based characterisation is known and has been used to show the tractability of many problems such as SAT [29], #SAT or #CQ for β-acyclic instances [9, 6]. The elimination order is based on the notion of nest points. In a hypergraph $H = (V, E)$, a *nest point* is a vertex $v \in V$ such that $E(v)$ is ordered by inclusion, that is, $E(v) = \{e_1, \dots, e_p\}$ with $e_1 \subseteq \cdots \subseteq e_p$. A *$\beta$-elimination order* $(v_1, \dots, v_n)$ *for* H is an ordering of V such that for every $i \leqslant n$, v_i is a nest point of $H \setminus \{v_1, \dots, v_{i-1}\}$. A closer inspection of the definition of β-elimination order $\prec$ shows that β-$\mathsf{fhow}(H, \prec) = \beta$-$\mathsf{how}(H, \prec) = 1$, showing that it corresponds to β-acyclicity. We can actually prove a more general result: the notion of β-acyclicity has been recently generalised by Lanzinger in [25] using a notion called nest sets. A set of vertices $S \subseteq V$ is a *nest set of* H if $\{e \setminus S \mid e \in E, e \cap S \neq \emptyset\}$ is ordered by inclusion. A *nest set elimination order* is a list $\Pi = (S_1, \dots, S_p)$ such that: $\bigcup_{i=1}^{p} S_i = 1$, $S_i \cap S_j = \emptyset$ and S_i is a nest set of $H \setminus \bigcup_{j<i} S_j$.

The width of a nest set elimination is $\mathsf{nsw}(H, \Pi) = \max_i |S_i|$ and the *nest set width* $\mathsf{nsw}(H)$ *of* H is defined to be the smallest possible width of a nest set elimination order of H. It turns out that our notion of width generalises the notion of nest set width, that is, we have β-$\mathsf{how}(H) \leqslant \mathsf{nsw}(H)$. More particularly, any order $\prec$ obtained from a nest set elimination order $\Pi = (S_1, \dots, S_p)$ by ordering each S_i arbitrarily verifies $\mathsf{nsw}(H, \Pi) \geqslant \beta$-$\mathsf{how}(H, \prec)$.

We summarise the above discussion in the following theorem:

▶ **Theorem 11** ($\star$)**.** *For every hypergraph $H = (V, E)$, we have: β-$\mathsf{htw}(H) \leqslant \beta$-$\mathsf{how}(H) \leqslant \mathsf{nsw}(H)$. In particular, H is β-acyclic iff β-$\mathsf{how}(H) = 1$.*

The goal of this paper is not to give a thorough analysis of β-fractional hyperorder width so we leave for future research several questions related to it: what is the complexity of computing the β-fractional hyperorder width of a hypergraph, how does it compare with other widths such as (incidence) treewidth, (incidence) cliquewidth, MIM-width or point-width [11]. For these measures of width, #SAT, a problem close to computing the number of answers in signed join queries, is known to be tractable (see [9] for a survey). We leave open the most fundamental question of comparing the respective powers of β-$\mathsf{fhtw}(\cdot)$ and β-$\mathsf{fhow}(\cdot)$. We also refer the interested reader to the full version where we discuss why the seemingly natural notion of β-hyperorder width has not appeared earlier in the literature.

Signed hyperorder width. In the case of signed join queries, one can deal with positive and negative atoms differently, which is not reflected by the definition of β-$\mathsf{fhow}(\cdot)$. We generalise these widths to signed hypergraphs by taking subhypergraphs only on the negative part, generalising a notion of acyclicity introduced by Brault-Baron in [4] that mixes β- and α-acyclicities for signed hypergraphs. Let $H = (V, E_+, E_-)$ be a signed hypergraph. Given an order $\prec$ on V, the *signed hyperorder width* $\mathsf{show}(H, \prec)$ *of* $\prec$ *wrt* H is defined as $\mathsf{show}(H, \prec) = \max_{E' \subseteq E_-} \mathsf{how}((V, E_+ \cup E'), \prec)$. The *signed hyperorder width* $\mathsf{show}(H)$ *of* H is defined as $\mathsf{show}(H) = \min_{\prec} \mathsf{show}((H, \prec)$. Fractional version of these widths could easily be defined but will not be needed in this paper. It is clear from the definition that if $E_+ = \emptyset$ then $\mathsf{show}(H, \prec) = \beta$-$\mathsf{how}(H, \prec)$ and if $E_- = \emptyset$, then $\mathsf{show}(H, \prec) = \mathsf{how}(H, \prec)$.

4.3 Complexity of exhaustive DPLL

The complexity of DPLL on a conjunctive query Q and order $\prec$ can be bounded in terms of the hyperorder width of $H(Q)$ wrt $\prec$:

▶ **Theorem 12.** *Let Q be a signed join query, $\mathbf{D}$ a database over domain D and $\prec$ an order on $\mathsf{var}(Q)$. Then $\mathsf{DPLL}(Q, \langle\rangle, \mathbf{D}, \prec)$ produces a $\succ$-ordered $\{\times, \mathsf{dec}\}$-circuit C of size $\mathcal{O}(\mathsf{poly}_k(|Q|)|\mathbf{D}|^{k+1})$ in time $\mathcal{O}(\mathsf{poly}_k(|Q|)|\mathbf{D}|^{k+1}\mathsf{polylog}|D|)$ such that $\mathsf{rel}(C) = [\![Q]\!]^{\mathbf{D}}$ where $k = \mathsf{fhow}(H(Q), \prec)$ if Q is positive and $k = \mathsf{show}(H(Q), \prec)$ otherwise.*

A full proof of Theorem 12 can be found in the full version. The proof is technical since there is nothing connecting the structure of the hypergraph of Q and the runtime of DPLL. Due to space constraints, we only give a short overview of the proof, trying to stress how both notions connect.

In this section, we fix a signed join query Q that has exactly one $\langle\rangle$-component (the case where Q has many $\langle\rangle$-component can be easily dealt with by constructing the Cartesian product of each $\langle\rangle$-component of Q), a database $\mathbf{D}$ and an order $\prec$ on $\mathsf{var}(Q) = \{x_1, \dots, x_n\}$ where $x_1 \prec \cdots \prec x_n$. We let D be the domain of $\mathbf{D}$, n be the number of variables of Q and m be the number of atoms of Q. To ease notation, we will write X instead of $\mathsf{var}(Q)$. For $i \leqslant n$, we denote $\{x_1, \dots, x_i\}$ by $X_{\preceq i}$. Similarly, $X_{\prec i} = X_{\preceq i} \setminus \{x_i\}$, $X_{\succ i} = \mathsf{var}(Q) \setminus X_{\preceq i}$ and $X_{\succeq i} = \mathsf{var}(Q) \setminus X_{\prec i}$. Finally, we let $\mathbf{R}_Q^{\mathbf{D}}$ be the set of (K, σ) such that $\mathsf{DPLL}(Q, \langle\rangle, \mathbf{D}, \prec)$ makes at least one recursive call to $\mathsf{DPLL}(K, \sigma, \mathbf{D}, \prec)$. The first thing to observe, is that thanks to the usage of a cache, the complexity of DPLL can naturally be stated as a function of $|\mathbf{R}_Q^{\mathbf{D}}|$:

▶ **Lemma 13** (⋆). *$\mathsf{DPLL}(Q, \langle\rangle, \mathbf{D}, \prec)$ produces a circuit of size at most $\mathcal{O}(|\mathbf{R}_Q^{\mathbf{D}}| \cdot |D| \cdot \mathsf{poly}(|Q|))$ in time $\mathcal{O}(|\mathbf{R}_Q^{\mathbf{D}}| \cdot \mathsf{poly}(|Q|) \cdot |D|\mathsf{polylog}|D|)$.*

Hence to prove Theorem 12, one only needs to bound the size of $\mathbf{R}_Q^{\mathbf{D}}$. To do that, we characterise the elements $(K, \sigma) \in \mathbf{R}_Q^{\mathbf{D}}$ in terms of hypergraph structure. The key notion we need is the notion of x-components, a definition akin to the one used in [9] to compile β-acyclic CNF formulas. Let $Q' \subseteq Q$ be a subquery of Q and x, y two variables of Q' such that $y \prec x$. An *x-path to y in Q'* is a list $x_0, a_0, \dots, x_p$ where $a_i \in \mathsf{atoms}(Q')$ is an atom of Q' on variables $\mathbf{x_i}$, x_i is a variable of $\mathbf{x_i}$, $x_0 = x$, $x_p = y$ and $x_i \preceq x$ for every $i \leqslant p$. Intuitively, it maps to a path in the hypergraph of Q' that starts from x and is only allowed to use vertices smaller than x. The *x-component of Q'* is the set of atoms a of Q' such that there exists an x-path to a variable y of a in Q'.

It turns out that the recursive calls of DPLL are x-components of some $Q' \subseteq Q$ and $x \in X$ where Q' is obtained from Q by removing negative atoms. Intuitively, these removed atoms are the ones that cannot be satisfied anymore by the current assignment of variables.

▶ **Lemma 14** (⋆). *Let $(K, \sigma) \in \mathbf{R}_Q^{\mathbf{D}}$ and let x be the biggest variable of K not assigned by σ. There exists τ a partial assignment of $X_{\succ x}$ such that $\tau|_{\mathsf{var}(K)} = \sigma$ and K is the x-component of $Q \Downarrow \tau$.*

Moreover, x-components are connected to the width notions from Section 4.2 as follows:

▶ **Lemma 15** (⋆). *Let Q be a signed join query on variables $X = \{x_1, \dots, x_n\}$, x_i a variable of Q and K_i its x_i-component. We let H be the hypergraph of Q, $H_1 = H$ and $H_{j+1} = H_j/x_j$. We have $N_{x_i}(H_i) = \mathsf{var}(K_i) \cap X_{\succeq x_i}$.*

Intuitively, Lemma 15 allows us to conclude that the variables from $X_{\succ x}$ of an x-component in a hypergraph H admit a fractional cover of value at most k where $k = \mathsf{fhow}(H, \prec)$. Now we illustrate how one can use it to get a bound on $\mathbf{R}_Q^{\mathbf{D}}$. We start with the case of positive conjunctive query. Let $(K, \sigma) \in \mathbf{R}_Q^{\mathbf{D}}$, σ. By Lemma 14 that there exists $\tau \supset \sigma$ such that K is the x-component of $Q \Downarrow \tau$, which is equal to Q in the case of positive conjunctive query. Moreover, τ must be compatible with every atom of Q, otherwise DPLL would have returned

$\perp$ already. In other words, τ satisfies every atom from the x-component of Q restricted to $X_{\succ x}$. But these atoms have a fractional cover of value at most k, hence by Theorem 2, there are at most $|\mathbf{D}|^k$ different τ for each x-component. Since there are at most n distinct x-component, a bound on $\mathbf{R}_Q^{\mathbf{D}}$ follows.

We can have a similar bound for the signed case but now $Q \Downarrow \tau$ is not Q anymore since some negative atoms may have been removed from Q because they are incompatible with τ. However, we know that the hypergraph of $Q \Downarrow \tau$ is obtained by removing negative atoms of Q, hence we know that the variables from $X_{\succ x}$ of any x-component of $Q \Downarrow \tau$ are covered by at most k atoms. Moreover, $Q \Downarrow \tau$ only contains atoms that are compatible with τ. Hence by Theorem 2 again, we can show that for a given K, there are at most $|\mathbf{D}|^k$ distinct τ such that $(K, \tau) \in \mathbf{R}_Q^{\mathbf{D}}$. Moreover, we can show that the possible K can be characterised by looking at the x-component of $Q \Downarrow \tau$ for every τ that is a solution of the join of at most k atoms projected on $X_{\succ x}$ which allows us to derive an $m^{k+1}|\mathbf{D}|^k$ bounds on $\mathbf{R}_Q^{\mathbf{D}}$ in this case. To wrap up, we can prove the following bounds on $\mathbf{R}_Q^{\mathbf{D}}$:

▶ **Lemma 16** ($\star$). *If Q is a positive join query with m atoms and n variables, $|\mathbf{R}_Q^{\mathbf{D}}| \leqslant n|\mathbf{D}|^k$ where $k = \mathsf{fhow}(H(Q), \prec)$. Otherwise $|\mathbf{R}_Q^{\mathbf{D}}| \leqslant nm^{k+1}|\mathbf{D}|^k$ where $k = \mathsf{show}(H(Q), \prec)$.*

Theorem 12 is a direct corollary of Lemmas 13 and 16. One may wonder why we do not use fractional width when Q contains negative atoms. The proof of Lemma 16 breaks in this case when we try to bound the number, for a given x, of x-component K that can appear in recursive calls. To prove Lemma 16, we bound it by taking at subset of at most k atoms of Q. To do it with fractional cover, one would need to consider every combination of atoms of Q having fractional cover at most k which we did not manage to bound by a polynomial in Q. We therefore leave this question open for future research but observe that it would give a complexity of at most $\mathcal{O}(2^m|\mathbf{D}|^{k+1}\mathsf{polylog}|D|)$ which is polynomial wrt data complexity.

5 Tractability results for signed join and conjunctive queries

Combining the algorithm for DA on ordered circuits from Section 3 with the algorithm of Section 4 gives tractability results on the complexity of direct access for signed join queries:

▶ **Theorem 17** ($\star$). *Given a signed join query Q, an order $\prec$ on $\mathsf{var}(Q)$ and a database $\mathbf{D}$ on domain D, we can solve the direct access problem for $\prec_{\mathsf{lex}}$ with precomputation $\mathcal{O}(|\mathbf{D}|^{k+1}\mathsf{polylog}|D| \cdot \mathsf{poly}_k(|Q|))$ and access time $\mathcal{O}(\mathsf{poly}(n) \cdot \mathsf{polylog}|D|)$ where $n = |\mathsf{var}(Q)|$ where $k = \mathsf{fhtw}(H(Q), \succ)$ if Q is positive and $k = \mathsf{show}(H(Q), \succ)$ otherwise.*

Negative join queries and #SAT. Theorem 17 generalises many tractability results from the literature. First of all, our result can directly be applied to #SAT, the problem of counting the number of satisfying assignment of a CNF formula. A CNF formula F with m clauses can directly be transformed into a negative join query Q_F with m atoms having the same hypergraph and into a database $\mathbf{D}_F$ on domain $\{0, 1\}$ and of size at most m such that $[\![Q_F]\!]^{\mathbf{D}_F}$ is the set of satisfying assignments of F. Indeed, a clause can be seen as the negation of a relation having exactly one tuple. For example, $x \vee y \vee \neg z$ can be seen as $\neg R(x, y, z)$ where R contains the tuple $(0, 0, 1)$. Hence, Theorem 17 generalises both [9] and [6] by providing a compilation algorithm for β-acyclic queries to any domain size and to the more general measure of β-hyperorder width. It also shows that not only counting is tractable but also the more general DA tasks. Theorem 17 also generalises the results of [25] which shows the tractability of the evaluation of negative join queries with bounded nest set width. Since a negative join query with nest set width k has β-hyperorder width at most

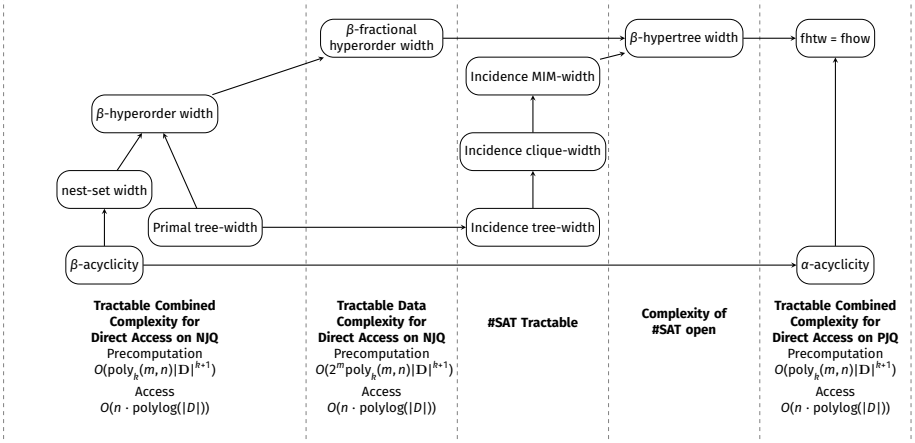


Figure 4 Landscape of hypergraph measures and known inclusions (depicted as an arrow) with tractability results for direct access on positive and negative join queries (PJQ/NJQ) and #SAT on CNF formulas. Here n is the number of variables, $\mathbf{D}$ the database, D the domain ($\{0,1\}$ for #SAT), m the number of atoms/clauses and k the width measure ($k = 1$ for α- and β-acyclicity).

k by Theorem 11, Theorem 17 implies that DA is tractable for the class of queries with bounded nest set width. In particular, counting the number of answers is tractable for this class, a question left open in [25].

Figure 4 summarises our contributions for join queries with negations and locates them in the landscape of known tractability results. Even if our result applies to signed conjunctive query, we summarise our contribution only for negative join queries and positive join queries since it allows to compare hypergraph measures (where tractability of signed queries is stated using signed hypergraphs parameters). The stated complexity are given assuming that a decomposition is provided in the input. While the complexity of computing the β-fractional hyperorder width is open, we observe that for nest set width, this decomposition could also be computed in FPT time in the size of the hypergraph [25]. In particular, it gives the following:

▶ **Theorem 18.** *Computing $\#[\![Q]\!]^{\mathbf{D}}$ can be computed in polynomial time when parametrized by nest-set width.*

Despite the fact that computing an optimal nest-set width elimination order is FPT, Theorem 18 only gives an XP algorithm for #SAT and not an FPT algorithm since the complexity of DPLL has a $O(m^k)$ dependency where m is the number of atoms.

Direct access for positive conjunctive queries. Theorem 17 allows to recover the tractability of DA for positive join queries with bounded fractional hypertree width proven in [12, 7]. Indeed, given an order $\prec$ on the vertices of a hypergraph, [7] introduces the notion of incompatibility number of $\prec$ which corresponds exactly to its fractional hyperorder width. Hence Theorem 17 implies the same tractability results for positive join query as [7, Theorem 10]. The complexity bounds from this paper are however better than ours and proven optimal since the preprocessing is of the form $\mathsf{poly}_k(Q)|\mathbf{D}|^k$ where we have $\mathsf{poly}_k(Q)|\mathbf{D}|^{k+1}$. We nevertheless believe that with a more careful analysis of the implementation of Algorithm 1, we could match this upper bound although this is not the focus of this paper. Another strong point of [12] (and also [8, Theorem 39] which is the arXiv version of [7]) is that it handles conjunctive queries, that is, join queries with projection which is not covered by Theorem 17. We demonstrate the versatility of the circuit-based approach by showing how one can also handle quantifiers directly on the circuit.

▶ **Theorem 19** (⋆). *Let C be a $\prec$-order circuit on domain D, variables $X = \{x_1, \ldots, x_n\}$ such that $x_1 \prec \cdots \prec x_n$ and $j \leqslant n$. One can compute in time $O(|C| \cdot \mathsf{poly}(n) \cdot \mathsf{polylog}(|D|))$ a circuit C' of size at most $|C|$ such that $\mathsf{rel}(C') = \mathsf{rel}(C)|_{\{x_1,\ldots,x_j\}}$.*

Now we can use Theorem 19 to handle conjunctive queries by first using Theorem 12 on the underlying join query to obtain a $\prec$-circuit and then by projecting the variables directly in the circuit. This approach works only when the largest variables in the circuits are the quantified variables. It motivates the following definition: given a hypergraph $H = (V, E)$, an elimination order $(v_1, \ldots, v_n)$ of V is S-connex if and only if there exists i such that $\{v_i, \ldots, v_n\} = S$. In other words, the elimination order starts by eliminating $V \setminus S$ and then proceeds to S. Given a conjunctive query Q and an elimination order $\prec$ on $\mathsf{var}(Q)$, we say that the elimination is free-connex if it is a $\mathsf{free}(Q)$-connex elimination order of $H(Q)$ where $\mathsf{free}(Q)$ are the free variables of Q[2]. We directly have the following:

▶ **Theorem 20** (⋆). *Given a conjunctive query $Q(Y)$, a free-connex order $\succ$ on $\mathsf{var}(Q)$ and a database $\mathbf{D}$ on domain D, we can solve direct access tasks for $\prec_{\mathsf{lex}}$ with precomputation $\mathcal{O}(|\mathbf{D}|^{k+1}\mathsf{polylog}|D| \cdot \mathsf{poly}_k(|Q|))$ and access time $\mathcal{O}(n \cdot \mathsf{polylog}(|D|))$ where $n = |\mathsf{var}(Q)|$ where $k = \mathsf{fhtw}(H(Q), \succ)$ if Q is positive and $k = \mathsf{show}(H(Q), \succ)$ otherwise.*

We observe that our notion of free-connex elimination order for Q is akin to [8, Definition 38] with two differences: first, in [8], it is allowed to only specify a preorder on $\mathsf{free}(Q)$ and the complexity of the algorithm is then stated with the best possible compatible ordering, which would be possible in our framework too. The second difference is that the orders are presented in reverse, that is, in their definition, the orders start with free variables and end with quantified variables. We decided to present free-connexity of elimination orders in this way so that it corresponds to the existing notion of free-connexity of tree decompositions. Now, Theorem 20 constructs a direct access for $\prec_{\mathsf{lex}}$ when $\succ$ is free-connex, so Theorem 20 proves the same tractability result as [8, Theorem 39], again with an extra $|D|$ factor but compatible with negative and signed conjunctive queries.

6 Future Work

Our new tractability results for solving DA tasks on signed conjunctive queries relies on a unifying framework for both positive and signed queries using factorised representation of the answer sets of the query. It opens many avenues for research. First, contrary to the positive query case, we do not yet have parameterised lower bounds on the preprocessing and access time needed for solving DA tasks on signed queries. Having a better understanding of what happens on the fractional relaxation of β-hyperorder width would be a first step toward proving such lower bounds. Also, we believe our analysis of the complexity of DPLL is not optimal and that with the right data structures, we should be able to prove an upper bound of the order $|\mathbf{D}|^{\mathsf{fhow}(Q)}$ instead of the $|\mathbf{D}|^{\mathsf{fhow}(Q)+1}$ for positive queries, hence matching the existing upper bounds exactly. We leave a more involved analysis of this algorithm for future work. Finally, we believe that the circuit representation we are using is promising for answering different kind of aggregation tasks and hence generalising existing results to the case of signed conjunctive queries. For example, FAQ and AJAR queries [24, 21] could be answered using this data structure by annotating the circuit with semi-ring elements and

[2] The notion of S-connexity already exists for tree decompositions. We use the same name here as the existence of an S-connex tree decomposition of (fractional) hypertree width k is equivalent to the existence of an S-connex elimination order of (fractional) hyperorder width k.

projecting them out as in Theorem 19. Similarly, we believe that the framework of [15] for solving DA tasks on conjunctive queries with aggregation operators may be generalised in a similar way to the class of ordered $\{\times, \mathsf{dec}\}$-circuits.

References

1 Guillaume Bagan. *Algorithmes et complexité des problèmes d'énumération pour l'évaluation de requêtes logiques. (Algorithms and complexity of enumeration problems for the evaluation of logical queries)*. PhD thesis, University of Caen Normandy, France, 2009. URL: https://tel.archives-ouvertes.fr/tel-00424232.

2 Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. Computing the jth solution of a first-order query. *RAIRO-Theoretical Informatics and Applications*, 42(1):147–164, 2008. doi:10.1051/ita:2007046.

3 Nurzhan Bakibayev, Tomáš Kočiskỳ, Dan Olteanu, and Jakub Závodnỳ. Aggregation and ordering in factorised databases. *Proceedings of the VLDB Endowment*, 6(14):1990–2001, 2013. doi:10.14778/2556549.2556579.

4 Johann Brault-Baron. *De la pertinence de l'énumération: complexité en logiques propositionnelle et du premier ordre*. PhD thesis, Université de Caen, 2013. URL: https://tel.archives-ouvertes.fr/tel-01081392.

5 Johann Brault-Baron. Hypergraph acyclicity revisited. *ACM Computing Surveys (CSUR)*, 49(3):1–26, 2016. doi:10.1145/2983573.

6 Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for beta-acyclic CNF-formulas. In *32nd International Symposium on Theoretical Aspects of Computer Science*, volume 30 of *LIPIcs*, pages 143–156. Schloss Dagstuhl, 2015. doi:10.4230/LIPIcs.STACS.2015.143.

7 Karl Bringmann, Nofar Carmeli, and Stefan Mengel. Tight fine-grained bounds for direct access on join queries. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 427–436, 2022. doi:10.1145/3517804.3526234.

8 Karl Bringmann, Nofar Carmeli, and Stefan Mengel. Tight fine-grained bounds for direct access on join queries. *arXiv preprint arXiv:2201.02401*, 2022. doi:10.48550/arXiv.2201.02401.

9 Florent Capelli. Understanding the complexity of #SAT using knowledge compilation. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–10. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005121.

10 Florent Capelli and Oliver Irwin. Direct access for conjunctive queries with negation. *CoRR*, abs/2310.15800, 2023. doi:10.48550/arXiv.2310.15800.

11 Clément Carbonnel, Miguel Romero, and Stanislav Živnỳ. Point-width and max-csps. *ACM Transactions on Algorithms (TALG)*, 16(4):1–28, 2020. doi:10.1145/3409447.

12 Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. *ACM Transactions on Database Systems*, jan 2023. doi:10.1145/3578517.

13 Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 393–409, 2020. doi:10.1145/3375395.3387662.

14 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 77–90, New York, NY, USA, 1977. ACM. doi:10.1145/800105.803397.

15 Idan Eldar, Nofar Carmeli, and Benny Kimelfeld. Direct access for answers to conjunctive queries with aggregation. *arXiv preprint*, 2023. doi:10.48550/arXiv.2303.05327.

16 Johannes K Fichte, Markus Hecher, Neha Lodha, and Stefan Szeider. An smt approach to fractional hypertree width. In *Principles and Practice of Constraint Programming: 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings 24*, pages 109–127. Springer, 2018. doi:10.1007/978-3-319-98334-9_8.

17 Robert Ganian, André Schidler, Manuel Sorge, and Stefan Szeider. Threshold treewidth and hypertree width. *Journal of Artificial Intelligence Research*, 74:1687–1713, 2022. doi:10.1613/jair.1.13661.

18 Georg Gottlob and Reinhard Pichler. Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width. *SIAM Journal on Computing*, 33(2):351–378, 2004. doi:10.1137/S0097539701396807.

19 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms (TALG)*, 11(1):4, 2014. doi:10.1145/2636918.

20 Jinbo Huang and Adnan Darwiche. DPLL with a Trace: From SAT to Knowledge Compilation. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 156–162, 2005. URL: http://ijcai.org/Proceedings/05/Papers/0876.pdf.

21 Manas R Joglekar, Rohan Puttagunta, and Christopher Ré. Ajar: Aggregations and joins over annotated relations. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 91–106, 2016. doi:10.1145/2902251.2902293.

22 Jens Keppeler. *Answering Conjunctive Queries and FO+MOD Queries under Updates*. PhD thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät, 2020. URL: http://edoc.hu-berlin.de/18452/22264.

23 Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. Faq: questions asked frequently. *arXiv preprint*, 2015. doi:10.48550/arXiv.1504.04044.

24 Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. Faq: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 13–28, 2016. doi:10.1145/2902251.2902280.

25 Matthias Lanzinger. Tractability beyond β-acyclicity for conjunctive queries with negation and sat. *Theoretical Computer Science*, 942:276–296, 2023. doi:10.1016/j.tcs.2022.12.002.

26 Dan Olteanu. Factorized databases: A knowledge compilation perspective. In *AAAI Workshop: Beyond NP*, 2016. URL: http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12638.

27 Dan Olteanu and Jakub Závodnỳ. Factorised representations of query results: size bounds and readability. In *Proceedings of the 15th International Conference on Database Theory*, pages 285–298. ACM, 2012. doi:10.1145/2274576.2274607.

28 Dan Olteanu and Jakub Závodnỳ. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):1–44, 2015. doi:10.1145/2656335.

29 S. Ordyniak, D. Paulusma, and S. Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theoretical Computer Science*, 481:85–99, 2013. doi:10.1016/j.tcs.2012.12.039.

30 Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 79:984–1001, sep 2013. doi:10.1016/j.jcss.2013.01.012.

31 Tian Sang, Fahiem Bacchus, Paul Beame, Henry A Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. *Theory and Applications of Satisfiability Testing*, 2004. URL: http://www.satisfiability.org/SAT04/programme/21.pdf.

32 Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, pages 3–18. ACM, 2016. doi:10.1145/2882903.2882939.

33 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases – Volume 7*, VLDB '81, pages 82–94. VLDB Endowment, 1981.