The Complexity of Querying Indefinite Data about Linearly Ordered Domains*

Ron van der Meyden
Department of Computer Science
Rutgers University, New Brunswick, NJ†

To appear in Journal of Computer and System Sciences

Abstract

In applications dealing with ordered domains, the available data is frequently indefinite. While the domain is actually linearly ordered, only some of the order relations holding between points in the data are known. Thus, the data provides only a partial order, and query answering involves determining what holds under all the compatible linear orders. In this paper we study the complexity of evaluating queries in logical databases containing such indefinite information. We show that in this context queries are intractable even under the data complexity measure, but identify a number of PTIME sub-problems. Data complexity in the case of monadic predicates is one of these PTIME cases, but for disjunctive queries the proof is non-constructive, using well-quasi-order techniques. We also show that the query problem we study is equivalent to the problem of containment of conjunctive relational database queries containing inequalities. One of our results implies that the latter is Π_2^p-complete, solving an open problem of Klug [JACM, 1988].

*This is a revised and extended version of a paper with the same title which appeared in the ACM Symposium on Principles of Database Systems, 1992.

†Author's current address: School of Computing Science, University of Technology, Sydney, PO Box 123, Broadway, NSW 2007, Australia.

1 Introduction

In applications dealing with ordered domains, the available data is frequently indefinite. While the domain is actually linearly ordered, only some of the order relations holding between points in the data are known. Thus, the data provides only a partial order, and query answering involves determining what holds under all the compatible linear orders. In this paper we study a class of logical databases containing such indefinite information.

An *indefinite order database* D consists of a set of ground atomic facts together with facts of the forms $u < v$ and $u \leq v$ asserting order relations between certain constants representing points in a linearly ordered domain. These *order constants* may be thought of as a special sort of null value, on which order constraints may be placed. We adopt an "open world semantics" for indefinite order databases, in which the relation '$<$' is interpreted over linear orders. With respect to this semantics, we establish bounds on the complexity of determining if the entailment relation $D \models \Phi$ holds, where Φ is a positive existential formula containing order relations.

Example 1.1: The following example illustrates the nature of query answering in indefinite order databases.

> A highly classified document is discovered to have been leaked during the night from the security compound at the US embassy in Moscow. There are no duplication facilities in the compound: the guilty party must have removed the document, copied it, and then replaced it. Thus the culprit was in the compound at least twice. The security guard's log shows agent S entering the compound, then leaving. Some time later, agent N is recorded entering. The guard's watch was broken, so exact times are not recorded. Worse, he confesses to having dozed off frequently during the night, so this is all the information his log shows. He is dishonourably discharged for dereliction of duty. Interrogation of agent S and agent N yields the following information: agent S admits to having been in the compound, and claims that while there, agent N also came into the compound. Agent S says he left before agent N did, but does not have a precise recollection of what times he entered and left. Agent N "takes the Fifth" and refuses to testify. This evidence does not appear to be much to go on, but it is enough to encourage the Internal Affairs officer to start further investigations into the activities of agents S and N: he has deduced that if the evidence is to be taken at face value, then one of the two was in the compound twice.

We may formalize this problem as follows. Let the predicate $IC(u, v, x)$ represent the fact that x was in the compound for a continuous period

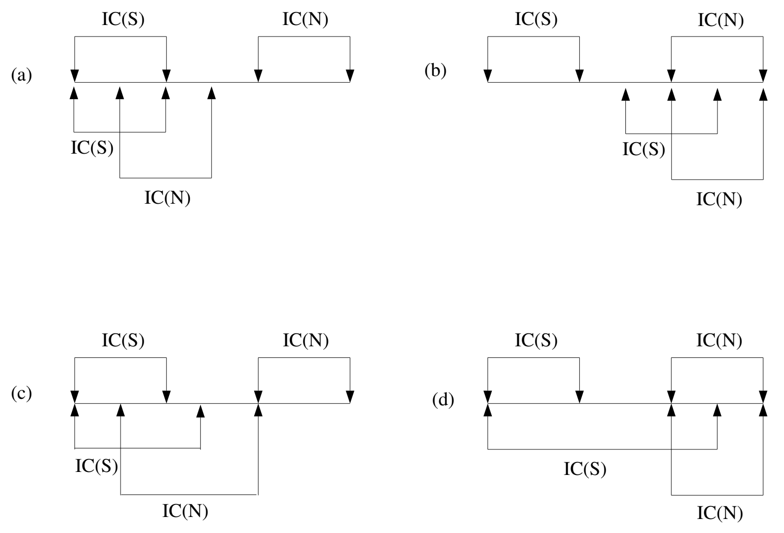


Figure 1: Some models of the data

starting at time u and ending at time v. Then the guard's log may be expressed as

$$IC(z_1, z_2, S), IC(z_3, z_4, N), z_1{<}z_2{<}z_3{<}z_4$$

Here $z_1 \dots z_4$ are order constants representing unknown points of time and the object constants S, N refer to agent S and agent N, respectively. The testimony of agent S may be represented as

$$IC(u_1, u_3, S), IC(u_2, u_4, N), u_1{<}u_2{<}u_3{<}u_4$$

where again the u_i are unknown order constants.

Suppose this set of facts has been entered in a knowledge base. Since time is a linearly ordered domain, the constraints on the constants z_i, u_i underdetermine the temporal relationships holding between these constants in any model of the data. For example, we could have any of the relationships $z_1{<}u_1$, $z_1 = u_1$, or $z_1 > u_1$ holding in models of the data. Thus, to obtain models of the data it is necessary to "topologically sort" the partial order in the data, adding additional constraints so as to obtain a linear order. Figure 1 shows some of the models resulting from this process. Here the top portion of each model derives from the guard's log, the bottom portion derives from the testimony of agent S. Note that distinct order constants may refer to the same point in the linear order, e.g., in model (a) $z_1 = u_1$.

We also need some integrity constraints: for example, the facts mentioned so far have a model (d) in which $z_1 = u_1$ and $z_2{<}u_3$, so that we have two overlapping, but not identical intervals representing periods for which agent S was in the compound. Clearly the intended semantics does not permit this. We need to eliminate models which have such overlapping but not identical intervals. Rather than incorporate such 'negative' information in the database, we will handle this by modifying queries. Thus, let Ψ be the formula

$$\exists x t_1 t_2 t_3 t_4 w[IC(t_1,t_2,x) \wedge IC(t_3,t_4,x) \wedge t_1{<}w{<}t_2 \wedge \\ t_3{<}w{<}t_4 \wedge (t_1{<}t_3 \vee t_2{<}t_4)]$$

which detects the condition we wish to eliminate. (This particular integrity constraint allows simultaneous departure and reentry.) The effect of the integrity constraint is then obtained by using the query $\Psi \vee \Phi$ in place of the query Φ, using the fact that $D \wedge \neg\Psi \models \Phi$ if and only if $D \models \Psi \vee \Phi$. The investigating officer may now reach his conclusion by noting that the formula $\Phi(x) = \exists t_1 t_2 t_3 t_4[IC(t_1,t_2,x) \wedge IC(t_3,t_4,x) \wedge t_1{<}t_3]$ expresses that x entered the compound at two distinct times t_1, t_3. Thus he may pose the query $\Psi \vee \Phi(S) \vee \Phi(N)$ ("Did either agent S or agent N enter the compound twice?") or, more generally, $\Psi \vee \exists x\Phi(x)$ ("Did someone enter the compound twice?"). We leave it to the reader to verify that both of these queries should be answered "yes". Note however, that the queries $\Psi \vee \Phi(S)$ and $\Psi \vee \Phi(N)$ should both fail (consider models (a) and (b)): there is not yet enough evidence for charges to be laid against either suspect. $\Box$

Many applications give rise to indefinite data about linear order. As in the example, the linearly ordered domain is often a time line. In the problem of seriation in archeology [16] each type of artifact is assumed to have been in use for some historical interval. Absolute data for these intervals is rarely available, but coincidence of two artifacts in a grave indicates that their intervals overlap. Golumbic [12] describes this and many other examples of indefinite order data in various domains, including behavioural psychology, biology, scheduling problems in operations research, and combinatorics.

Indefinite order information also arises naturally in a variety of contexts in artificial intelligence. Allen [2] has pointed out that in natural language most temporal reports describe relations between intervals, rather than give absolute times. He proposes an algebra based on thirteen primitive temporal relations between intervals, such as "Interval I overlaps interval J" and gives an algorithm for making (incomplete) inferences about interval relations. Another example of indefinite order data in artificial intelligence is *nonlinear planning* [28]. Here, rather than the solution to a planning problem being a linear sequence of actions, one constructs a partially ordered set of actions. This allows some flexibility in the order of execution of actions, and is also able to express concurrently executable plans. However, it is still necessary to reason about the compatible linear orders, since these correspond to the possible executions of the plan.

We have stated the problem we consider in terms of indefinite information, but it is closely related to optimization problems for relational database queries containing inequalities, which have been studied by Klug [17]. A query $Q1$ is said to be *contained* in a query $Q2$ if in every relational database (containing only definite information) the answer (set of tuples) to $Q1$ is a subset of the answer to $Q2$. Klug noted that the classical homomorphism theory [5] for containment of conjunctive queries, which shows that this problem is NP complete, does not extend to queries containing inequalities (although it does work for a subset of these queries, the semi-interval queries.) He was able to provide an upper bound of Π_2^p, but no lower bound. We establish in Section 2 a reduction from query processing in indefinite order databases to query containment. Using this reduction, one of our results (Theorem 3.3) provides the missing lower bound of Π_2^p-hard.[1]

There exists a substantial literature on reasoning about constraints, which includes a number of studies of complexity issues. In the database field, the problem of inferring inequalities from other inequalities has been studied in connection with predicate locking [27]. The more general problem of inferring linear inequalities of the form $ax + by + \ldots \leq p$ from other such inequalities has been considered in the context of applications to constraint logic programming [21, 29].

The complexity of reasoning about order relations has also received some attention in the AI literature. Vilain et al. [33] study the complexity of inferring relationships between intervals in Allen's interval algebra, that is, they study the complexity of determining whether an interval relationship $I_1 R I_2$ follows, where R is a relation in the interval algebra. They show that even this problem has NP hard complexity. As a remedy they propose to restrict the expressiveness of the data to a point based language with relations '$<$', '$\leq$' and '$\neq$'. The problem of deriving point relationships, that is, the problem of determining if uRv follows for $R \in \{<, \leq, \neq\}$, has polynomial time complexity [30, 3]. Golumbic and Shamir [11] present a finer grained analysis of the complexity of inferring interval relations in the interval algebra, showing the effect on complexity of various restrictions on the set of primitive relations.

However, as indicated by Example 1.1, queries about the possible relationships between points or intervals are only a very restricted subset of the queries one might wish to ask a database with indefinite data about linear order: a minimal class would seem to be the positive existential queries, built using existential quantification, disjunction and conjunction from atomic formulae which involve proper predicates as well as the order predicates '$<$' and '$\leq$'. While much is already known about the complexity of querying indefinite databases containing null values subject to '$\neq$' constraints only [32, 1], there does not appear to have been any analysis of the complexity of this more general class of queries in indefinite order databases, aside from the related work of Klug already mentioned. Our contribution in this paper is to provide such an analysis.

We consider the following measures of complexity, introduced by Vardi [31].

[1]Kanellakis et al. [15] have previously shown that containment of conjunctive queries using quadratic equation constraints is Π_2^p-complete. They also consider data complexity for a notion of (definite) constraint databases different from ours, in which constraints occur as conditions on universally quantified tuples.

Combined complexity is the complexity of the set $\{\langle D, \Phi\rangle | D \models \Phi\}$. This measures complexity of query answering as a function of both the size of the query and of the database. In most cases, this is not a realistic measure of complexity in database applications, since a database is generally many orders of magnitude larger than a query. *Data complexity* takes this into account by factoring out the size of the query, which may be presumed to be small. The data complexity of a (fixed) query Φ is the complexity of determining membership in the set $\{D \mid D \models \Phi\}$. Note that this captures the optimal complexity of any algorithm for answering Φ, but ignores the cost of producing this algorithm, i.e., the cost of compilation. This means that data complexity results should be interpreted with some care. As we will see in Section 6, it is sometimes possible to show that data complexity is low without knowing how to compile queries to run efficiently. For this reason we take some pains to develop combined complexity results, since these may provide the only practicable algorithms. For completeness we also consider *expression complexity.* The expression complexity of a database D is the complexity of the set $\{\Phi \mid D \models \Phi\}$.

Our results show that even the data complexity of very simple forms of queries is intractable. Thus, further constraints are required to obtain tractable inference problems. We consider a number of different parameters, and provide a characterization of the classes of problems stated in terms of these parameters that have polynomial time complexity. One of the constraints we consider is severe: it is the restriction that predicates be monadic. While monadic predicates are insufficiently expressive to represent the interval data required in many applications, this restriction is nevertheless of interest.

Example 1.2: In gene alignment in biology one wishes to compare a number of sequences of bases, represented by symbols C, G, A and T, for evidence that they are related. Mutations correspond to insertions and deletions of fragments in such sequences, so it is necessary to permit "gaps" when doing the comparison: see Figure 2 for a sample alignment. The space of possible alignments may be represented by an indefinite order database in which C, G, A and T are monadic predicates and a sequence $s_1 s_2 \ldots s_n \in \{C, G, A, T\}^*$ corresponds to the facts

$$s_1(u_1), s_2(u_2), \ldots, s_n(u_n), u_1{<}u_2{<}\ldots{<}u_n.$$

Using the query modification method for integrity constraints used in Example 1.1 one may represent various restrictions on the acceptable alignments: disjoining $\exists t[A(t) \wedge G(t)]$ disallows the alignment of A and G (This constraint is violated at the leftmost position of the alignment in Figure 2). Algorithms for query answering may then be used to answer the question "does there exist an alignment of the sequences which satisfies the integrity constraints?" □

Certain queries using n-ary predicates also reduce to the monadic case. Suppose, for example, that all predicates have a single "temporal" argument that may participate in order relations. That is, we are interested in facts

$$GCAAGTCGATCGAG$$
$$AC--GTCG-GCGA-$$

Figure 2: Aligning two sequences

holding at time points rather than over intervals. Then queries which quantify only over time reduce to the monadic case. For example, consider the query

$$\exists uv[P(a,u) \wedge u{<}v \wedge P(b,v)]$$

expressing that there exists a time when $P(a)$ holds and a later time when $P(b)$ holds. This query may be answered using algorithms for the monadic case by treating the expressions $P(a,x)$ and $P(b,x)$ as (distinct) monadic predicates on the variable x.

Surprisingly, even the restriction to monadic predicates is not enough to obtain tractable combined complexity. Also, while data complexity under this constraint is tractable, the proof is non-trivial. Thus, although the monadic case has very limited expressive power from the point of view of applications, its non-triviality makes it of theoretical interest. Moreover, we feel that an understanding of the monadic case is a prerequisite for finding tractable classes of problems with higher arity. It is therefore worthwhile to investigate additional constraints.

One constraint that leads to tractability in the monadic case is a bound on the *width* of the database. Width is a parameter of the partial order that the data imposes on the order constants. Informally, it measures the maximum number of "concurrent" order constants, which potentially refer to the same point in the linear order. For example, suppose that the database is a record of the reports of a number of agents independently observing the world. If each provides a linearly ordered set of observations, then the width of the database is the number of agents. The database of Example 1.1 has width two. A bound on the width of the database appears to be a reasonable constraint for some applications.

We also consider a constraint on queries in the monadic case. A conjunctive query is said to be *sequential* if its order variables are linearly ordered by its order atoms. For example, the query

$$\exists xyz[x{<}y{<}z \wedge P(x) \wedge Q(x) \wedge P(y) \wedge Q(z)]$$

is sequential. Sequential queries ask: "does a particular sequence of events occur?"

We now summarize the results presented in the paper. We begin with a study of the connections between the three different consequence relations $\models_{\mathcal{O}}$, obtained by choosing for the class $\mathcal{O}$ of linear orders either the class **Fin** of finite linear orders, the class **Z** of linear orders isomorphic to the integers, or the class **Q** of dense linear orders isomorphic to the rationals. We establish

predicate arity	complexity type		
	data	expression	combined
n-ary	co-NP complete	NP complete	Π_2^p complete
monadic	PTIME	PTIME	co-NP complete

Table 1: Complexity of query problems: n-ary predicates

query type	database width	
	bounded	unbounded
sequential	PTIME	PTIME
non-sequential	PTIME	co-NP complete

Table 2: Combined complexity of conjunctive queries: monadic predicates

polynomial-time reductions that enable us to transfer both upper and lower bounds for the case $\mathcal{O} = \mathbf{Fin}$ to the other two. (Thus, all the results we state hold under any of these semantics.)

We then turn to an analysis of complexity for the semantics with $\mathcal{O} = \mathbf{Fin}$. In their general forms, the query problems considered turn out to be (probably) intractable, as indicated by the first row of Table 1. Here each entry gives a complexity class for which the corresponding query problem is complete. For example, the entry in the first row of the first column indicates that (1) every query has data-complexity in co-NP and (2) there exists a query with co-NP hard data-complexity. In each case the lower bound result can be established using conjunctive queries and the upper bound result holds for disjunctive queries as well as conjunctive queries.

The lower bound results require at least binary relations. As indicated by the second row of Table 1, we do obtain an improvement by restricting to monadic predicates. Conjunctive monadic queries may be shown to have PTIME data complexity by a straightforward "greedy" algorithm. However, for disjunctive queries the proof of PTIME data complexity is non-trivial: we present for the disjunctive case a *non-constructive* proof, using well-quasi-order techniques, which yields the fact that the data complexity is in PTIME, but without explicitly describing for each query a specific algorithm which solves the problem with that complexity.

The proof that conjunctive queries using monadic predicates have co-NP hard combined complexity involves non-sequential queries and databases of unbounded width. However, if we restrict either to sequential queries or to databases of width bounded by a constant, combined complexity may be shown to be in PTIME, as indicated in Table 2.

We actually give two algorithms showing that conjunctive monadic queries have PTIME combined complexity over databases of bounded width. One works only for conjunctive queries, and merely decides entailment; the other is less efficient, but may be modified to yield a procedure which either declares that the query is entailed, or enumerates all the (minimal) models of the database in which the query is false, with no more than a polynomial amount of time between outputs. (The latter approach also works for disjunctive queries, but has exponential complexity in the number of disjuncts.)

The structure of the paper is as follows. Section 2 considers the relations between a number of different semantics for indefinite order databases, depending on the type of the linear order. Section 3 is concerned with upper and lower bounds for queries containing predicates of arbitrary arity. The remainder of the paper studies the monadic case. Section 4 deals with upper and lower bounds for conjunctive monadic queries. A number of the results developed in this section are crucial to the proofs in later sections. Section 5 deals with expression and combined complexity in the disjunctive monadic case. Section 6 is devoted to the non-constructive proof that data complexity of disjunctive monadic queries is in PTIME. In Section 7 we briefly consider the generalization of indefinite order databases obtained by admitting inequality constraints. Section 8 concludes by discussing further work.

2 Definitions and Preliminary Results

This section is devoted to setting up the semantic framework for order databases by formally defining three consequence relations, depending on the structure of the linear order in models. We establish reductions between these three relations that permit us to focus on just one type of semantics, the finite model semantics, and give a technical characterization of this semantics that will be helpful in establishing complexity results.

We work with a two sorted first order language, containing a sort of *objects*, as usual, as well as an *order sort*, representing points in a linearly ordered domain. Thus, we require that the arguments of predicates be typed, and that the occurrence of constants and variables respect the typing. The language contains no function symbols. Atomic formulae are of one of two kinds:

1. *proper atoms* of the form $P(\mathbf{a})$, where P is a predicate and $\mathbf{a}$ is a tuple of constants or variables of the appropriate sort, or

2. *order atoms* of the form $u < v$ or $u \leq v$, where u and v are order constants or variables.

An *indefinite order database* D consists of a finite set of ground atoms of either variety. Queries will be positive existential sentences of the first order language based on the proper predicates and the relation '$<$'. That is, queries are constructed from proper atoms and order atoms using only the operators '$\wedge$', '$\vee$' and '$\exists$'. A query that does not contain '$\vee$' is said to be *conjunctive*. For the purposes of complexity analysis we assume queries are in disjunctive normal form, i.e., are disjunctions of conjunctive queries. (However, for brevity we will sometimes write queries in non-normal form: these are to be understood as denoting equivalent disjunctive normal form queries.)

We will be concerned with restricted classes of databases and queries in which just one of the order relations '$<$', '$\leq$' may appear. When a result applies to such a restricted case this will be indicated by prefixing the word 'database' or 'query' by the set of relations permitted. For example, "$\{<\}$-databases" refers to databases in which the relation '$\leq$' does not occur.

A structure for an order database D will be a (two-sorted) first-order structure M in which the relation '$<$' denotes a linear order $<_M$ on the order sort, and in which $u \leq v$ is interpreted as $u < v \vee u = v$. Such a structure will be a model of a database just in case it supports the database as a first order theory. We reserve the word 'points' to refer to elements of the order sort; elements of the object sort will be called 'objects'. We do not make a unique names assumption [26] in this paper: distinct constants may refer to the same point. For order constants this is because we explicitly wish to allow distinct order constants to refer to the same point; for object constants the adoption of the unique names assumption would have no effect on query entailment, since we deal only with positive existential queries.

It is convenient to adopt a simplifying assumption: queries will be assumed not to contain constants. By a well known construction, there is no loss of generality in this. We may introduce a new monadic predicate P_u for each constant symbol u, and add the facts $P_u(u)$ to the database. Then the query $\Phi(u)$ containing the constant u is equivalent to the query $\exists t[P_u(t) \wedge \Phi(t)]$ in which the constant has been eliminated. The advantage of this construction is that it enables us to discard from models the mappings interpreting constants, when determining satisfaction of a query. This will be important in some of our proofs. This assumption also simplifies the treatment of equality.

We will consider various semantics for databases, corresponding to different restrictions on the linear order. If $\mathcal{O}$ is a class of linear order types we define

$$Mod_{\mathcal{O}}(D) = \{M \mid M \models D \text{ and } <_M \text{ is of type } \mathcal{O}\}.$$

The class $\mathcal{O}$ will be either the class **Fin** of finite linear orders, the class $\mathbf{Z}$ of linear orders isomorphic to the natural numbers or the class $\mathbf{Q}$ of dense linear orders isomorphic to the rationals. For each class $\mathcal{O}$ we obtain a consequence relation $\models_{\mathcal{O}}$ defined by

$$D \models_{\mathcal{O}} \Phi \quad \text{iff} \quad M \models \Phi \text{ for all } M \in Mod_{\mathcal{O}}(D).$$

For the restricted form of database and query we are considering, these consequence relations are closely related, as we now show.

First, we need a standard model theoretic notion [6]. A *homomorphism* from a model M to a model M' is a mapping h from the domain of M to the domain of M' such that

- If a is of sort object (order) then $h(a)$ is of sort object (order).
- If the interpretation of constant u in M is a, then the interpretation of u in M' is $h(a)$.
- For all relations P, including order relations, and all elements a_i of the domain of M, if $P(a_1, \ldots, a_n)$ holds in M then $P(h(a_1), \ldots, h(a_n))$ holds in M'.

It is well known that if there exists a homomorphism from M to M' then for every positive existential query Φ, if $M \models \Phi$ then $M' \models \Phi$.

Proposition 2.1: The following containments hold between the consequence relations: $\models_{\mathbf{Fin}} \subseteq \models_{\mathbf{Z}} \subseteq \models_{\mathbf{Q}}$.

Proof: We show that $D \models_{\mathbf{Z}} \Phi$ implies $D \models_{\mathbf{Q}} \Phi$, by proving the contrapositive. Suppose that $D \not\models_{\mathbf{Q}} \Phi$. Then there exists a model $M \in Mod_{\mathbf{Q}}(D)$ with $M \not\models \Phi$. Let S be the image under the interpretation mapping of the constants of D in M. Add additional elements of the order domain of M to S so that the points in S, with the order induced from M, comprise an order isomorphic to $\mathbf{Z}$. Now let M' be the restriction of the model M to the resulting subset of the domain. Clearly there exists a homomorphism from M' to M, from which $M' \not\models \Phi$ follows. Hence $D \not\models_{\mathbf{Z}} \Phi$ also, since $M' \in Mod_{\mathbf{Z}}(D)$. This shows that $D \models_{\mathbf{Z}} \Phi$ implies $D \models_{\mathbf{Q}} \Phi$. A similar argument shows that $D \models_{\mathbf{Fin}} \Phi$ implies $D \models_{\mathbf{Z}} \Phi$. $\square$

To see that these consequence relations are inequivalent, observe that $\models_{\mathbf{Z}} \exists t_1 t_2 [t_1 < t_2]$ but not $\models_{\mathbf{Fin}} \exists t_1 t_2 [t_1 < t_2]$, since **Fin** contains the linear order consisting of a single point. Similarly, note that if $D = \{P(u), P(v), u < v\}$ and

$$\Phi = \exists t_1 t_2 t_3 [P(t_1) \wedge t_1 < t_2 < t_3 \wedge P(t_3)]$$

then $D \models_{\mathbf{Q}} \Phi$ but not $D \models_{\mathbf{Z}} \Phi$. In both of these examples we have variables which occur only in order atoms. This is in fact a necessary condition for such examples. Say that a query is *tight* if in each disjunct, every variable occurs in some proper atom. (Thus, none of the queries in the present paragraph is tight, since in each the variable t_2 appears in no proper atom.) Then we have the following:

Proposition 2.2: If Φ is a tight query then $D \models_{\mathbf{Fin}} \Phi$ iff $D \models_{\mathbf{Z}} \Phi$ iff $D \models_{\mathbf{Q}} \Phi$.

Proof: By Proposition 2.1 it suffices to show that $D \models_{\mathbf{Q}} \Phi$ implies $D \models_{\mathbf{Fin}} \Phi$. We establish the contrapositive. Suppose that there exists $M \in Mod_{\mathbf{Fin}}(D)$ with $M \not\models \Phi$, where the order domain is the set $\{0 \ldots n\}$. Modify M by enlarging the order domain to the set of rational numbers. This produces a model M' in $Mod_{\mathbf{Q}}(D)$. Suppose that $M' \models \Phi$ and let θ be a satisfying assignment for the variables of Φ. Since Φ is tight, for each variable t we must have $\theta(t)$ in the set $\{0 \ldots n\}$. But this implies $M \models \Phi$, a contradiction. Thus $M' \not\models \Phi$, establishing that $D \not\models_{\mathbf{Q}} \Phi$. $\square$

In many applications, queries will generally be tight. For example, consider relational database queries containing inequalities. In formulating such queries we are generally interested in comparing data values retrieved from the database. This means that we are only interested in values actually occurring in some database relation. The result shows that for these purposes the three types of semantics are equivalent.

However, it is sometimes natural to write non-tight queries. We have already seen an instance of this in Example 1.1, in which we used the query

$$\exists x t_1 t_2 t_3 t_4 w[IC(t_1,t_2,x) \wedge IC(t_3,t_4,x) \wedge t_1{<}w{<}t_2 \wedge \\ t_3{<}w{<}t_4 \wedge (t_1{<}t_3 \vee t_2{<}t_4)]$$

to express the integrity constraint that overlapping intervals of the form $IC(u,v,x)$ must be identical. Note that the occurrence of w in this query is not tight. Therefore it is of some interest to understand the relations between the three semantics for non-tight queries. We will establish polynomial time reductions of the relations $\models_{\mathbf{Q}}$ and $\models_{\mathbf{Z}}$ to the relation $\models_{\mathbf{Fin}}$. These reductions enable query answering procedures for one semantics to be used for another, with no more than a polynomial loss in complexity. We note that these reductions will be established in one direction only, so they do not serve to transfer lower bounds on complexity from the relation $\models_{\mathbf{Fin}}$ to the relations $\models_{\mathbf{Q}}$ and $\models_{\mathbf{Z}}$. The reason we do not need to establish the converse reductions is that all of the lower bounds to be proved for $\models_{\mathbf{Fin}}$ will make use of tight queries only, so these bounds apply to all three semantics, by Proposition 2.2.

We begin with the reduction for $\models_{\mathbf{Z}}$. Suppose that the query Φ contains n distinct variables. Given a database D, let $l_1,\ldots,l_n,r_1,\ldots,r_n$ be $2n$ new constants. Add to D the atoms $l_1{<}l_2{<}\ldots{<}l_n$ and $r_1{<}r_2{<}\ldots{<}r_n$, as well as $l_n{<}u{<}r_1$ for each order constant u of D, and call the resulting database D'.

Proposition 2.3: For every database D, $D \models_{\mathbf{Z}} \Phi$ if and only if $D' \models_{\mathbf{Fin}} \Phi$.

Proof: We first show that $D' \models_{\mathbf{Fin}} \Phi$ implies $D \models_{\mathbf{Z}} \Phi$. Suppose $D' \models_{\mathbf{Fin}} \Phi$ and let $M \in Mod_{\mathbf{Z}}(D)$. Consider the finite model M' obtained from M by restricting the domain to the image of the constants in D. Add n points $u_1 < \ldots < u_n$ less than the least point in M' and another n points $v_1 < \ldots < v_n$ greater than the greatest point in M', and interpret l_i as u_i and r_i as v_i, for $i = 1 \ldots n$. The resulting structure M'' is a clearly a finite model of D'. Since $D' \models_{\mathbf{Fin}} \Phi$, we must have $M'' \models \Phi$. There exists a homomorphism from M'' to M, so it follows that $M \models \Phi$ also. This establishes that $D' \models_{\mathbf{Fin}} \Phi$ implies $D \models_{\mathbf{Z}} \Phi$.

For the converse, suppose that $D \models_{\mathbf{Z}} \Phi$. Let M' be a finite model of D'. By restricting the domain to the image of the constants of D' in M', restricting the proper facts to just those needed to support D, and renaming elements of the order domain, we obtain a finite model M'' of D' such that

1. The order domain of M'' is the set $\{-n,\ldots,k+n\}$.

2. The order constants of D are interpreted by M'' in the set $\{0,\ldots,k\}$.

3. The constant l_i is interpreted in M'' as $-i$, and the constant r_i is interpreted as $k+i$, for $i = 1 \ldots n$.

4. There exists a homomorphism from M'' to M'.

Let M be the model obtained from M'' by extending the order domain to $\mathbf{Z}$. Since $D \models_{\mathbf{Z}} \Phi$ there exists a satisfying assignment θ for Φ in the model M. Consider the order variables V of Φ that θ maps outside of the set $\{0, \ldots, k\}$. By construction of M, none of these variables can occur in a proper atom in Φ, hence they occur only in order atoms. There are at most n of these variables. Hence, by changing the assignment θ on V, we may construct an assignment θ' that maps the variables V into the set $\{-n, \ldots, k+n\}$ without changing the order relationships that hold. That is, for any order variables u, v occurring in Φ, $\theta'(u) < \theta'(v)$ if and only if $\theta(u) < \theta(v)$. It then follows that $M'' \models \Phi$, hence also that $M' \models \Phi$. This establishes that $D \models_{\mathbf{Z}} \Phi$ implies $D' \models_{\mathbf{Fin}} \Phi$. $\Box$

It is also possible to give a reduction of the semantics based on the rationals to finite models. This reduction involves showing that over the rationals, every non-tight query is equivalent to some tight query. It is convenient to introduce some auxiliary notions first. We begin by describing a normal form for databases and queries which simplifies some of our results. Consider the following rules, which may be used to transform a database or conjunctive query. By considering each disjunct independently, they may also be used to transform a disjunctive query.

N1. If there exist atoms $u_1 \leq u_2$, $\ldots$, $u_{n-1} \leq u_n$ and $u_n \leq u_1$, then identify $u_1, \ldots, u_n$.

N2. Delete any atom of the form $u \leq u$.

When applying rule N1 to transform a query we replace all occurrences of $u_2, \ldots, u_n$ by u_1 and delete the quantifiers for $u_2, \ldots, u_n$. (Recall that we have assumed that queries do not contain constants.)

The rules N1 and N2 are valid in the sense that if D is transformed to D' and Φ is transformed to Φ' then $D \models \Phi$ iff and only if $D' \models \Phi'$. This is clear for rule N2. Validity of rule N1 when applied to a query is also clear, since $u_1 \leq u_2 \leq \ldots \leq u_n \leq u_1$ is equivalent to $u_1 = u_2 = \ldots = u_n$. Validity of N1 when transforming a database relies on the fact that queries do not contain constants. We will say that a database or conjunctive query is *normalized* if it is invariant under application of rules N1 and N2. Since the transformations are valid and terminate, we may assume henceforth that all databases and queries are normalized.

Now, we introduce the notion of the *order graph* associated with a database or *conjunctive* query. This is the directed graph whose vertices are the order constants of D, or the order variables of the query Φ, respectively. For each atom $u < v$ in the database or query there is an edge from u to v labelled '$<$', and for each atom $u \leq v$ there is an edge from u to v labelled '$\leq$'.

A *normalized* database or conjunctive query is inconsistent if and only if its order graph contains a cycle. For, by rules N1 and N2 there can be no cycles containing only edges labelled '$\leq$'. On the other hand, if there exists a cycle containing an edge labelled '$<$', then there exists a constant (or variable) u such that the inconsistent atom $u < u$ is entailed. Conversely, if the order graph is acyclic (that is, if it is a directed acyclic graph, or *dag*) then the database or query is consistent. This is simply the well known fact that it is possible to "topologically sort" a dag, that is, produce a linear order satisfying all the order relations entailed by the dag. We will assume throughout that we are dealing only with consistent databases and queries.

Note that in topologically sorting an order dag, in the usual sense of the term, we in fact satisfy all '$\leq$' edges by means of *strict* inequalities. We will use the phrase "topological sort" of an order dag to refer to a slightly more general class of compatible linear orders than usually meant by this term. For us, a topological sort will be any mapping f from the vertices of the dag *onto* a linearly ordered set, such that f preserves the order relations. Such mappings may be constructed by the following nondeterministic procedure.

If there is no edge to a vertex in a dag we say it is *minimal.* The usual topological sorting process proceeds by repeatedly selecting minimal vertices. Instead, we define a vertex v of an order dag to be *minor* if there does not exist an ascending path ending in that vertex which passes through an edge labeled '$<$'. The sorting procedure will operate in a number of stages. At each stage we have a partially constructed linear order, together with a subgraph of the original graph, which contains all the vertices not yet mapped into the linear order. Initially the linear order is empty and the graph of unsorted vertices is the original graph. We repeat the following steps until the entire graph has been sorted. First, we non-deterministically select some set S of vertices, subject to the constraints

S1. Each element of S is minor in the subgraph of unsorted vertices.

S2. If $u \in S$ and there is an edge from v to u labelled '$\leq$' then $v \in S$.

We map the elements of S to the 'next' point of the finite linear order being constructed, and delete the vertices S from the graph of unsorted vertices.

Example 2.4: Suppose we are given the normalized set of order atoms $u < v < w, u \leq t \leq w$. The minor vertices of the corresponding graph are u and t. We demonstrate one of the topological sorts of this graph. We begin with an empty linear order and the original set of atoms. The minimal unsorted vertices at this stage are u and t. Let us choose $S = \{u, t\}$ as the elements mapping to the first point x_1 of the linear order. Deleting the elements in S leaves the graph $v < w$. This has one minor vertex v, so we now must take $S = \{v\}$ as the set of vertices mapping to the next point x_2 of the linear order. Deleting these vertices leaves just the vertex w, which we map to the last point x_3 of the linear order. Thus the topological sort obtained consists of the linear order with three points $x_1<x_2<x_3$, together with the mapping f with $f(u) = f(t) = x_1$, $f(v) = x_2$ and $f(w) = x_3$. Other topological sorts of this order are obtained by making different choices for the set of minor elements S. □

To see that the nondeterministic algorithm indeed results in a mapping satisfying the order relations, note that if the database contains an atom $u < v$, then v will not be a minor vertex until u has been deleted. Hence u will be mapped to the linear order first, and $u < v$ will be satisfied. For a constraint $u \leq v$, note that if v is an element of S and u has not yet been mapped to the linear order, then by rule S2 the vertex u is an element of S also, so the constraint will be satisfied with $f(u) = f(v)$.

Conversely, every mapping f from an order graph onto a linear order which preserves the order relations can be obtained as a result of the topological sorting procedure. To show this, note that we may choose at the k-th stage the set $S = \{v | f(v) = t_k\}$, where t_k is the k-th element of the linear order. It is straightforward to verify that these sets satisfy constraints S1 and S2.

The order atoms of a database need not correspond to all the order relations between order constants that may be inferred from the database. For example, if we have atoms $u < v$ and $v \leq w$ then we may infer $u < w$. The following rules may be used to add such derived atoms to a database (or query).

1. If u and v are *distinct* variables and there exists in the graph a path from u to v then add the atom $u \leq v$.

2. If there exists in the graph a path from u to v through an edge labelled '$<$' then add the atom $u < v$.

If a database or conjunctive query is closed under these rules then we will say that it is *full*. A disjunctive query is full if each disjunct is full. For example, the query

$$\exists uvw[Q(u,v,w) \wedge u \leq w] \vee \exists uvw[Q(u,v,w) \wedge u \leq v \wedge v \leq w \wedge u \leq w]$$

is full, but the query $\exists uvw[Q(u,v,w) \wedge u \leq v \wedge v \leq w]$ is not, since it does not contain the derived atom $u \leq w$. It is not difficult to show that every query is equivalent to some full query.

We may now establish the reduction for the dense order semantics. Let Φ be a full query. Delete from each disjunct of Φ any order variables that occur only in order atoms in that disjunct, (in other words, those that do not occur in any proper atom) as well as any quantifiers or order atoms containing those variables, and call the resulting query Φ'. For example, if Φ is the full query

$$\exists uvw[P(u,w) \wedge u \leq v \wedge v \leq w \wedge u \leq w]$$

in which the variable v does not occur in any proper atom then Φ' is the query $\exists uw[P(u,w) \wedge u \leq w]$ in which all order atoms containing this variable have been deleted.

Lemma 2.5: If Φ is a full query then $D \models_{\mathbf{Q}} \Phi$ if and only if $D \models_{\mathbf{Q}} \Phi'$, for every database D.

Proof: We assume without loss of generality that Φ is consistent. It is obvious that $D \models_{\mathbf{Q}} \Phi$ implies $D \models_{\mathbf{Q}} \Phi'$. For the converse, assume that $D \models_{\mathbf{Q}} \Phi'$ and let M be any model in $Mod_{\mathbf{Q}}(D)$. Then there exists a satisfying assignment θ' for some disjunct Ψ' of Φ'. Let S be a set of order constraints over variables U. Given a subset V of U, let T be the set of constraints in S which involve only the variables V. It is known [3] that if S is full then any assignment of V satisfying T may extended to an assignment of U satisfying S.[2] Since the variables deleted from Ψ occur only in order atoms, it follows from this that θ' may be extended to a satisfying assignment θ of the corresponding disjunct Ψ of Φ. □

Corollary 2.6: For every full query Φ and database D we have $D \models_{\mathbf{Q}} \Phi$ if and only if $D \models_{\mathbf{Fin}} \Phi'$

Proof: Follows directly from Proposition 2.2 and Lemma 2.5 on noting that the query Φ' is tight. □

Propositions 2.3 and 2.2 and Corollary 2.6 show that it suffices to restrict attention to finite models. In fact we may use an even smaller class of models, the *minimal models*, which are just those models obtained from the atoms of the database by interpreting the object constants as themselves, and interpreting the order constants by topologically sorting the graph of the database. We write $Mod(D)$ for this class of models and let $\models$ be the corresponding consequence relation.

Example 2.7: Let D be the database consisting of the order atoms $u < v < w$, $u \leq t \leq w$ from Example 2.4 together with the proper atoms $B(a,t)$, $B(b,w)$, in which a and b are object constants. Then one minimal model is the model with object domain $\{a,b\}$ and order domain consisting of the three points $x_1 < x_2 < x_3$. The object constants are interpreted as themselves, and the order constants are interpreted by the mapping f with $f(u) = f(t) = x_1$, $f(v) = x_2$ and $f(w) = x_3$ obtained from the topological sort of Example 2.4. The atomic facts holding in the model are $B(a,x_2), B(b,x_3)$. □

We now establish a result which explains why we use the term "minimal model" to refer to the models obtained by topologically sorting a database. The set of all models (with any type of linear order) may be (quasi) ordered by $M \leq N$ when there exists a homomorphism from M to N. The following states that the minimal models of a database are in fact minimal in this order.

Proposition 2.8: For every model N (of any order type) of a database D there exists a minimal model M with a homomorphism $h : M \to N$.

[2]This result is closely related to Fourier's method [7] for elimination of variables from linear inequality constraints of the form $a_1X_1 + \ldots + a_nX_n \leq b$.

Proof: Let the model N interpret the object constants of D by the function f and the order constants by the function g. For the object domain of the model M we take the set of object constants of D, with each object constant interpreted as itself. For the order domain of M we take the image of the function g, with each order constant u interpreted as $g(u)$. The linear order on the order domain of M is that induced from N. Finally, the proper atoms holding in M are just those that are the image under the interpretation of the constants of some proper atom of D. Let h be the function that maps an element a of the object domain of M (i.e., a constant of D) to $f(a)$, and acts as the identity function on the order domain. It is straightforward to verify that this is a homomorphism, and that M is in fact a minimal model. □

Corollary 2.9: For every database D and query Φ we have $D \models \Phi$ if and only if $D \models_{\mathbf{Fin}} \Phi$.

Proof: Immediate from Proposition 2.8. □

The reductions of this section show that even if we are interested in integer and rational order it suffices to restrict our attention to the finite model semantics. For the remainder of the paper we will develop our results with respect to this semantics. However, it is technically more convenient to work with minimal models, so we will state and prove our results in terms of the consequence relation $\models$. Corollary 2.9 provides the justification for this.

The following parameter of databases will be important in the sequel. Let U be the set of vertices of a dag G. A subset A of U is an *anti-chain* of G if there does not exist in G a path from any vertex $u \in A$ to another vertex $v \in A$. The *width* of G is the maximum cardinality of an antichain of G. The width of a normalized database or conjunctive query is the width of the associated dag.

If for two order constants u, v in a database the set $\{u, v\}$ is an anti-chain, then they may be viewed as being potentially concurrent. In models of the database any of the relations $u < v$, $u = v$ or $v < u$ may hold. Intuitively, the width of a database is a measure of the extent of indefiniteness at each stage of a topological sort of the database. For example, a database recording the reports of k observers, each providing a linear sequence of events, has width k. We will see below that width of databases is an important parameter in the complexity of query processing. Broadly speaking, query processing has lower complexity in databases with bounded width.

We comment that if D is a database of width k then we may assume that no vertex in the graph of D has more than $2k$ successors. For, given a vertex u, all atoms of the form $u \leq v$ are redundant except those with v minimal in the subgraph of D generated by $\{w | D \models u \leq w\}$. There can be at most k such minimal vertices. A similar argument applies to atoms of the form $u < v$. The number $2k$ is optimal, as may be seen from the database $D = \{u \leq v_i \mid i = 1 \ldots k\} \cup \{v_i \leq w_i \mid i = 1 \ldots k\} \cup \{u < w_i \mid i = 1 \ldots k\}$.

We now discuss a connection between the indefinite information query problem we study in this paper and a problem arising in the optimization of queries in relational databases [17, 30]. A relational database may be viewed as a finite structure for sorted first order logic. In case a linear order is of interest this may be treated as an additional relation of the structure - unlike the data relations we allow this to be infinite. This means that relational databases with order are just instances of models of indefinite order databases with finite object domains.

A *relational conjunctive query with inequalities* is an expression Q of the form $\{\mathbf{x} : \Phi(\mathbf{x}, \mathbf{y})\}$ where $\mathbf{x}$ and $\mathbf{y}$ are disjoint sequences of variables and $\Phi(\mathbf{x}, \mathbf{y})$ is a conjunction of proper and order atoms with variables among $\mathbf{x}, \mathbf{y}$. The answer set $Ans(Q, M)$ of a conjunctive query Q in a database M is the set of tuples $\langle \mathbf{a} \rangle$ of appropriately sorted elements of the domains of M such that $M \models \exists \mathbf{y} \Phi(\mathbf{a}, \mathbf{y})$. In the degenerate case that the sequence of variables $\mathbf{x}$ is empty we take the answer set to be the set containing only the null tuple $\langle\rangle$ if $M \models \exists \mathbf{y} \Phi(\mathbf{y})$, and the empty set otherwise.

If Q_1 and Q_2 are two relational queries then Q_1 is said to be *$\mathcal{O}$-contained in Q_2* if for all relational databases M in which the order relation is of type $\mathcal{O}$ we have $Ans(Q_1, M) \subseteq Ans(Q_2, M)$. Testing for containment allows for the optimization of conjunctive queries by the elimination of redundant atoms. The following shows that containment is at least as hard as query answering in indefinite order databases.

Proposition 2.10: Combined complexity of conjunctive queries in indefinite order databases with respect to $\models_{\mathcal{O}}$ is equivalent under PTIME reductions to $\mathcal{O}$-containment of relational conjunctive queries with inequalities.

Proof: Given an indefinite order database D and a conjunctive query Φ, we construct queries Q_1 and Q_2 such that Q_1 is contained in Q_2 if and only if $D \models \Phi$. If D is the set of atoms $\{A_1, \ldots, A_n\}$ take $Q_1 = \{\langle\rangle \mid A_1 \wedge \ldots \wedge A_n\}$, and take $Q_2 = \{\langle\rangle \mid \Phi\}$. It is plain that if M is a database with order of type $\mathcal{O}$ then $M \models D$ iff $\langle\rangle \in Ans(Q_1, M)$ and that $M \models \Phi$ iff $\langle\rangle \in Ans(Q_2, M)$. Since $\langle\rangle$ is the only potential answer to these queries, it is immediate from the definitions that $D \models_{\mathcal{O}} \Phi$ iff Q_1 is $\mathcal{O}$-contained in Q_2. (We use the fact that, by an argument similar to that for Corollary 2.9 the relation $D \models_{\mathcal{O}} \Phi$ is the same whether it is defined with respect to models with finite object domains or object domains with arbitrary cardinality.)

Conversely, suppose we are given queries $Q_1 = \{\mathbf{x} : \Phi_1(\mathbf{x}, \mathbf{y})\}$ and $Q_2 = \{\mathbf{x} : \Phi_2(\mathbf{x}, \mathbf{z})\}$. Define the database D to contain the atoms in the conjunction $\Phi_1(\mathbf{a}, \mathbf{b})$ where the $\mathbf{a}$ and $\mathbf{b}$ are fresh constants of the appropriate sorts, and define Φ to be $\exists \mathbf{z} \Phi_2(\mathbf{a}, \mathbf{z})$. It is straightforward to show that Q_1 is $\mathcal{O}$ contained in Q_2 if and only if $D \models_{\mathcal{O}} \Phi$. □

We will use this result to obtain a lower bound on containment of relational conjunctive queries with inequalities in the next section.

3 Databases with n-ary Predicates

In this section we begin our study of the complexity of query problems in indefinite order databases, establishing both upper and lower bounds for these problems in their most general form. These bounds will show that queries are intractable under each of the three types of complexity measure. The rest of the paper will be devoted to a search for natural restrictions under which complexity decreases. In this section, and for the remainder of the paper, we confine ourselves to results for the finite model semantics.

Upper bounds for the finite model semantics follow directly from the observation in the previous section that it suffices to restrict attention to minimal models. It is clear that the non-deterministic process constructing the minimal models operates in a polynomial number of steps. Thus, noting that positive existential queries have expression complexity in NP with respect to first order structures, we have the following immediate consequences of Corollary 2.9. (By means of the reduction of Proposition 2.10, Part (1) is equivalent to a bound on query containment previously noted by Klug [17].)

Proposition 3.1: (1) The combined complexity of indefinite order databases and positive existential queries is in Π_2^p.
(2) The data complexity of positive existential queries in indefinite order databases is in co-NP.
(3) The expression complexity of indefinite order databases with respect to positive existential queries is in NP.

This result holds for disjunctive as well as conjunctive queries. We now set about showing that these bounds can be met by corresponding lower bounds stated in terms of conjunctive queries.

Theorem 3.2: There exists a conjunctive $\{<\}$-query containing only binary predicates with co-NP hard data complexity on $\{<\}$-databases.

Proof: The proof is by reduction from monotone 3–satisfiability [10]. We show that there exists a query Φ and a polynomial time reduction from sets of monotone 3–clauses $\mathcal{S}$ to $\{<\}$-databases D such that $D \models \Phi$ if and only if $\mathcal{S}$ is unsatisfiable. Given object constants a, b, c and order constants u, v, w, t define the database $D(a, b, c; u, v, w, t)$ to be the set of atoms

$$\{P(u,a), P(u,b), u{<}v, P(v,a), P(v,c), v{<}w, P(w,b), P(w,c),\\ P(t,a), P(t,b), P(t,c)\}$$

depicted in Figure 3. Let $\varphi(x)$ be the query

$$\exists t_1 t_2 t_3 [P(t_1, x) \wedge t_1{<}t_2 \wedge P(t_2, x) \wedge t_2{<}t_3 \wedge P(t_3, x)]$$

also shown in Figure 3. The combination of this database and query "expresses" the disjunction "a or b or c" in the following sense.

D1. In every model of $D(a, b, c; u, v, w, t)$ either $\varphi(a)$ or $\varphi(b)$ or $\varphi(c)$ holds.

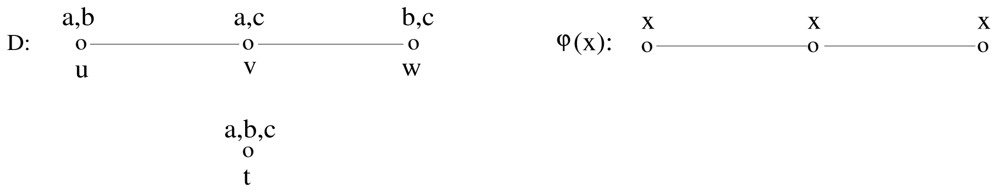


Figure 3: Simulating Ternary Disjunctions

D2. There exists a model in which $\varphi(a)$ is true but $\varphi(b)$ and $\varphi(c)$ are false. (Namely, the model in which $t = w$.) Similarly, there exist models in which only $\varphi(b)$ is true, or in which only $\varphi(c)$ is true.

Suppose we are given a set S of positive 3–clauses over a set of propositional letters L. The simulation of ternary disjunctions just described will be used to construct a $\{<\}$-database $D(S)$ and a query $\psi(x)$ for which the set $\{l \in L \mid M \models \psi(l)\}$ simulates, as M varies over minimal models of $D(S)$, the set of valuations satisfying the set S. One apparent way to do this is to let the database contain a component of the form $D(l_1, l_2, l_3; u, v, w, t)$ for each clause $l_1 \vee l_2 \vee l_3$. Unfortunately, propositional letters may occur in more than one clause and this may result in interference among the components. Instead, we will generate disjunctions independently and then transmit these disjunctions to the propositional letters. Specifically, for the i-th clause $l_{i,1} \vee l_{i,2} \vee l_{i,3}$ in S we introduce *new* object constants a_i, b_i, c_i and order constants u_i, v_i, w_i, t_i, and let D_i be the set of facts

$$D(a_i, b_i, c_i; u_i, v_i, w_i, t_i) \cup \{Q(l_{i,1}, a_i), Q(l_{i,2}, b_i), Q(l_{i,3}, c_i)\}.$$

Note that we treat the propositional letters l as object constants. Define $D(S)$ to be the union of the databases D_i, and let $\psi(x)$ be the query $\exists t[Q(x,t) \wedge \varphi(t)]$.

We can do the same for a set S' of negative clauses, using complemented constants $\overline{l}$ and facts $\left\{Q\left(\overline{l_{i,1}}, a_i\right), Q\left(\overline{l_{i,2}}, b_i\right), Q\left(\overline{l_{i,1}}, c_i\right)\right\}$ instead. Take F to be the set of facts of the form $Comp\left(l, \overline{l}\right)$ for l in the set L of propositional letters. Then given a set S of positive 3–clauses and a set S' of negative 3-clauses, we claim that $D \models \Phi$ exactly when the set of clauses $S \cup S'$ is unsatisfiable, where $D = D(S) \cup D(S') \cup F$ and $\Phi = \exists xy(\psi(x) \wedge Comp(x,y) \wedge \psi(y))$.

To see this, assume first that $D \models \Phi$, and suppose V is a valuation of L which satisfies $S \cup S'$. Then for each clause $l_{i,1} \vee l_{i,2} \vee l_{i,3}$ in S there exists an index j such that $V(l_{i,j}) = 1$. By the construction above we may choose a model M_i of D_i such that $M_i \models \psi(l_{i,k})$ only if $k = j$, and similarly for the negative clauses. Composing these models, we obtain a model M of D with the property that $M \models \psi(l)$ implies $V(l) = 1$ and $M \models \psi\left(\overline{l}\right)$ implies $V(l) = 0$. But since M is a model of D, we must have $D \models \Phi$. This is readily seen to imply that there exists a propositional letter l such that both $V(l) = 1$ and $V(l) = 0$, a contradiction.

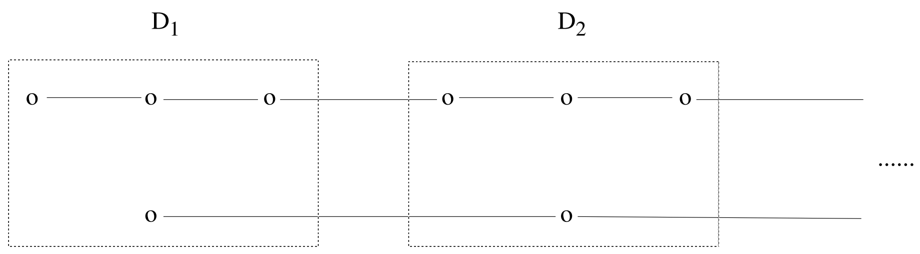


Figure 4: A width two database

Conversely, suppose that M is a model of D with $M \not\models \Phi$. We construct a valuation V as follows: if $M \models \psi(l)$ we put $V(l) = 1$, else $V(l) = 0$. We claim this is a satisfying valuation of $S \cup S'$. For the positive clauses, this follows directly from the construction of the D_i, since there must exist an index j such that $M \models \psi(l_{i,j})$. Consider next the negative clauses $\overline{l_1} \vee \overline{l_2} \vee \overline{l_3}$. If this clause is not satisfied, then $V(l_1) = V(l_2) = V(l_3) = 1$, which together with the fact that M falsifies the query, implies that $M \not\models \psi\left(\overline{l_i}\right)$ for each i. But this contradicts the fact that D was constructed so as to ensure that in each model at least one of the $\psi\left(\overline{l_i}\right)$ is satisfied. $\square$

We note that it is possible to make the database of this proof have bounded width. The "disjunction-generating" parts of the D_i are the only components in this proof containing order constants, and the proof still goes through if these are placed in a linear order as depicted in Figure 4, yielding a database of width two. Furthermore, the proof does not depend on the fact that the constants a_i, b_i, c_i and the constants $l_i, \overline{l_i}$ are object constants: it still goes through if we take these to be order constants instead. Indeed, we may also place all of these constants in a linear sequence following one of the sequences of Figure 4: this way the width of the database is still two.

It is possible to establish the same lower bound also with respect to $\{\leq\}$-databases and $\{\leq\}$-queries. The proof is almost identical, except that we use for the basic "disjunction generating" components the databases $D(u, v, w)$ consisting of all atoms $P(x, y, z)$ such that (x, y, z) is a permutation of the sequence of order constants (u, v, w). In this case we use the query

$$\varphi(x) = \exists yz[P(x, y, z) \wedge x \leq y \leq z]$$

where now x is an order variable. (This requires that we change the type of a number of variables and constants in the proof of Theorem 3.2.) It is readily verified that this query and database satisfy the properties D1 and D2 upon which the proof of Theorem 3.2 relies. As before, the proof may be modified to yield a database $D(S)$ of bounded width: in this case a bound of three is required.

The proof of the next result uses the following complete problem: A Π_2 formula of quantified propositional logic is an expression

$$\forall p_1 \ldots p_n \exists q_1 \ldots q_m[\alpha] \tag{1}$$

where α is a formula of propositional logic containing only the variables $p_1 \ldots p_n q_1 \ldots q_m$. Such a formula is true if for every assignment of boolean truth value to the variables p_i there exists an assignment of truth values to the variables q_i under which the formula α is true. The set Π_2-SAT is the set of all true Π_2 formulae. This is a generalization of the problem of satisfiability to the polynomial hierarchy. It is known that the set Π_2-SAT is complete for the level Π_2^p of this hierarchy [4].

Theorem 3.3: The combined complexity of $\{<\}$-databases and conjunctive $\{<\}$-queries is Π_2^p hard.

Proof: We use a reduction from Π_2-SAT. We reuse some ideas from the previous proof to express binary disjunctions. Consider the database

$$D_i = \{P_i(u_i, t), P_i(v_i, f), u_i{<}v_i, P_i(w_i, t), P_i(w_i, f)\}$$

As before, we may use the formula

$$\varphi_i(x) = \exists t_1 t_2 [P_i(t_1, x) \wedge P_i(t_2, x) \wedge t_1{<}t_2]$$

to express the disjunction $\varphi_i(t) \vee \varphi_i(f)$. We will use the databases D_i and the formulae $\varphi_i(x)$ to simulate the assignment of truth values to the variables p_i. To simulate the calculation of the truth value of the formula α, we let E be the set of facts

	$And(t,t,t)$	$Or(t,t,t)$	
$Istrue(t)$	$And(t,f,f)$	$Or(t,f,t)$	$Not(t,f)$
	$And(f,t,f)$	$Or(f,t,t)$	$Not(f,t)$
	$And(f,f,f)$	$Or(f,f,f)$	

Now we define inductively the query $Val(\alpha, \mathbf{z}, x)$ where α is a formula of propositional logic in the propositional variables $p_1 \ldots p_k$ and $\mathbf{z}$ is the vector of variables $z_1 \ldots z_k$. Intuitively this asserts that the truth value of the formula α under the assignment $z_1 \ldots z_k$ to the variables $p_1 \ldots p_k$ is x. If α is a propositional variable p_i then $Val(\alpha, \mathbf{z}, x)$ is the formula $x = z_i$. For the inductive case we have

$$\begin{aligned}
Val(\neg\alpha, \mathbf{z}, x) &= \exists t [Not(t, x) \wedge Val(\alpha, \mathbf{z}, t)] \\
Val(\alpha \wedge \beta, \mathbf{z}, x) &= \exists t_1 t_2 [And(t_1, t_2, x) \wedge Val(\alpha, \mathbf{z}, t_1) \wedge Val(\beta, \mathbf{z}, t_2)] \\
Val(\alpha \vee \beta, \mathbf{z}, x) &= \exists t_1 t_2 [Or(t_1, t_2, x) \wedge Val(\alpha, \mathbf{z}, t_1) \wedge Val(\beta, \mathbf{z}, t_2)]
\end{aligned}$$

Note that at each level we need to use fresh existentially quantified variables. A straightforward induction shows that if each z_i is either the constant t or f, representing truth and falsity of the propositional constant p_i, respectively, then the database E entails $Val(\alpha, \mathbf{z}, x)$ if and only if x is the truth value of the formula α under the assignment $\mathbf{z}$. (The use of equality in the definition of the operator Val is purely for convenience. Strictly, we have not permitted equality in our query language, but it is straightforward to eliminate. For example, $Val(\neg p_1, z_1, x)$ is the formula $\exists t(Not(t, x) \wedge t = z_1)$ which is equivalent to $Not(z_1, x)$.)

Now encode the quantified boolean formula (1) using the query

$$\begin{aligned}\Phi = \exists z_1 \ldots z_n[&\varphi_1(z_1) \wedge \ldots \wedge \varphi_n(z_n) \wedge \\ &\exists x z_{n+1} \ldots z_{n+m}\{Istrue(x) \wedge Val(\alpha, z_1 \ldots z_{n+m}, x)\}]\end{aligned}$$

and let the database D be the union of the databases D_i and E. We claim that $D \models \Phi$ exactly when the quantified boolean formula is true. For, suppose that the formula is true, and let M be any model of D. By construction of the D_i the model M supports either $\varphi_i(t)$ or $\varphi_i(f)$. The truth of Φ in M now follows from the truth of the quantified boolean formula and the meaning of the formula Val. Conversely, suppose Φ is true in every model of D. By construction, there exists for every vector $z_1 \ldots z_n$ of truth values a model M of D such that $M \models \varphi_i(x)$ if and only if $x = z_i$. Since M supports Φ there exist truth values $z_{n+1} \ldots z_{n+m}$ such that the formula α is true under the assignment $z_1 \ldots z_{n+m}$. This shows that the quantified formula (1) is true. □

We note that by means of the reduction of Proposition 2.10, we obtain from Theorem 3.3 a lower bound of Π_2^p hard for containment of relational conjunctive queries, resolving an open problem of Klug [17].

As before, it is also possible to prove the lower bound using only $\{\leq\}$-databases and $\{\leq\}$-queries. We also note that it is possible to modify the proof to use only a fixed finite set of binary predicates instead of the infinite set of predicates P_i. One way to do this is to use a chain of facts $P(u, v, u_0), R(u_0, u_1), R(u_1, u_2), \ldots, R(u_{i-1}, u_i), Q(u_i)$ of length i instead of the atom $P_i(u, v)$ in the database, and use

$$\exists t_0 \ldots t_i[P(x, y, t_0) \wedge R(t_0, t_1) \wedge \ldots \wedge R(t_{i-1}, t_i) \wedge Q(t_i)]$$

in the query in place of each occurrence $P_i(x, y)$. We may then make all predicates binary by means of the well-known reduction of n-ary predicates to binary.

Theorem 3.4: There exists an indefinite order database with NP hard expression complexity for conjunctive queries.

This follows from the fact that already relational databases have NP hard expression complexity for conjunctive queries. A proof of this is implicit in the proof of Theorem 3.3: if α is a propositional formula containing propositional variables $x_1, \ldots, x_n$ then the query

$$\exists x \exists z_1 \ldots z_n[Istrue(x) \wedge Val(\alpha, z_1 \ldots z_n, x)]$$

is entailed by the set E just in case the formula α is satisfiable.

4 Conjunctive Monadic Queries

We now embark on our study of restricted forms of the query problems with complexity lower than the general case, which we have just seen to be probably intractable. The lower bounds of Section 3 all required the use of at least binary predicates. As we argued in Section 1, there are applications for which monadic predicates suffice, so we are led to investigate the case in which all predicates are monadic. In this section we focus on conjunctive queries. The following sections will deal with the disjunctive case. Some of the ideas we develop in this section are crucial to later results.

We have seen that the combined complexity of order databases is Π_2^p-complete when we have binary predicates. It will emerge in this section that the restriction to monadic predicates does not suffice to reduce this to a polynomial time bound. Therefore, we investigate what further natural restrictions are required to achieve this reduction in complexity. A bound on the width of the database will be shown to be one restriction that suffices. Another, orthogonal case involves a restriction on the query: a certain class of conjunctive queries, sequential queries, will be shown to have polynomial time complexity.

When all predicates are monadic, we may confine our attention to predicates in which the single argument is an order argument. This is because there can now be no interaction between order arguments and object arguments. Any conjunctive query containing only monadic proper predicates can be written in the form $\exists \mathbf{x}[\Phi_1(\mathbf{x})] \wedge \exists \mathbf{t}[\Phi_2(\mathbf{t})]$ in which the first component contains all and only those parts of the query concerning objects. More precisely, the variables $\mathbf{x}$ are all object variables, the variables $\mathbf{t}$ are all order variables, the query Φ_1 contains only proper atoms whose single argument is of type object, and the query Φ_2 contains no proper predicates with object arguments. That is, Φ_2 contains only order atoms and proper atoms with order arguments. Since the component $\exists \mathbf{x}[\Phi_1(\mathbf{x})]$ involves no order variables it does not interact with indefiniteness in the database and may be directly evaluated, in time $O(nlog\ n)$, where n is the number of predicates, against the definite proper facts in the database. Thus the main source of complexity in the query is the component $\exists \mathbf{t}[\Phi_2(\mathbf{t})]$, which contains no predicates with object arguments. (Object predicates may also be eliminated from disjunctive queries by applying the above argument to each disjunct.)

Once we discard object constants, a very useful way to understand normalized monadic databases is as vertex labelled versions of the dags defined in Section 2, in which we label vertices by one or more predicate symbols. If u is an order constant we write $D[u]$ for the set of predicates P such that D contains the atom $P(u)$. All of these predicates label the corresponding vertex in the dag. Normalized conjunctive queries Φ may similarly be interpreted as labelled dags. In this case the vertices are the order variables of Φ, and we write $\Phi[t]$ for the set of predicates P such that Φ contains the atom $P(t)$. For example, if Φ is the query

$$\exists t_1 t_2 t_3 t_4 [P(t_1) \wedge Q(t_1) \wedge P(t_2) \wedge R(t_3) \wedge S(t_4) \wedge t_1 < t_2 < t_3 \wedge t_2 \leq t_4]$$

then we have $\Phi[t_1] = \{P, Q\}$ and $\Phi[t_4] = \{S\}$. The dag of this query is depicted in Figure 5. Here solid lines represent edges labelled '$<$' and broken lines indicate edges labelled '$\leq$'. Clearly there is a one to one correspondence be-

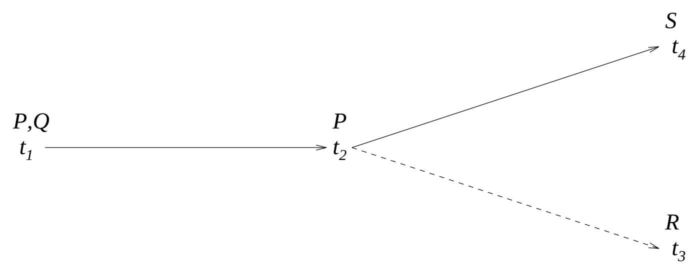


Figure 5: The dag associated with a query

tween monadic databases or queries and their labelled dag representations: we will therefore sometimes describe monadic queries and databases by presenting diagrams of their dags.

A query Φ will be said to be *sequential* if it is a conjunctive query of the form

$$\exists t_1 \ldots t_n [t_1 r_1 t_2 \wedge t_2 r_2 t_3 \wedge \ldots \wedge t_{n-1} r_{n-1} t_n \wedge \Psi(t_1, \ldots, t_n)]$$

where Ψ is quantifier free and contains no order relations, and each r_i is either '$<$' or '$\leq$'. Note that sequential queries have graphs of width one.

Sequential queries, finite models, and monadic databases of width one may be perspicuously represented as words over a special alphabet. Given a set *Pred* of monadic predicates, let $A = \mathcal{P}(Pred)$ be the power set of *Pred*. We define the set $FW(Pred) = A \cdot (\{<, \leq\} \cdot A)^*$ of *flexi-words* over *Pred* to be the set of all finite sequences of the form

$$a_1 r_1 a_2 r_2 \ldots r_{n-1} a_n$$

where for all i, $a_i \in A$ and $r_i \in \{<, \leq\}$. Then the sequential query Φ may be represented by the flexi-word $\Phi[t_1] r_1 \Phi[t_2] r_2 \ldots r_{n-1} \Phi[t_n]$. Conversely, to each element of $FW(Pred)$, there corresponds a sequential query, unique up to renaming of variables, as well as a database of width one, unique up to renaming of constants. We will switch at our convenience between these representations. For example, if p and q are flexi-words, we may write $p \models q$: here p is to be interpreted as a database and q is to be interpreted as a sequential query. (Here we exploit the fact that queries contain no constants.) The representation of finite models as flexi-words is similar. That is, a finite model M with order domain consisting of the points $u_1 < u_2 < \ldots < u_n$ corresponds to the flexi-word $M[u_1] < M[u_2] < \ldots < M[u_n]$.

If Φ is a conjunctive monadic query then a *path* in Φ is a maximal sequential subquery of Φ. In terms of the labelled dag representation, a path of a query corresponds to a labelled subgraph of its dag which has width one, and is maximal with this property. Thus the paths of the query of Figure 5 are the queries

$$\exists t_1 t_2 t_4 [P(t_1) \wedge Q(t_1) \wedge P(t_2) \wedge S(t_4) \wedge t_1 < t_2 \leq t_4]$$

and

$$\exists t_1 t_2 t_3 [P(t_1) \wedge Q(t_1) \wedge P(t_2) \wedge R(t_3) \wedge t_1 < t_2 < t_3]$$

corresponding to the flexi-words $\{P,Q\} < \{P\} \leq \{S\}$ and $\{P,Q\} < \{P\} < \{R\}$. We write $Paths(\Phi)$ for the subset of $FW(Pred)$ corresponding to paths of Φ. Similarly, if D is a database then we may define $Paths(D)$ to be the set of flexi-words corresponding to the maximal linear databases contained in D.

Lemma 4.1: If D is a monadic database and Φ is a conjunctive monadic query then $D \models \Phi$ if and only if $D \models p$ for every path $p \in Paths(\Phi)$.

Proof: It is clear that if $D \models \Phi$ then D entails every path of Φ. For the converse, suppose that $D \models p$ for every path $p \in Paths(\Phi)$. We show by induction on the size of the query Φ that $D \models \Phi$. The base case, where Φ is the empty query, is trivial. Suppose that M is an arbitrary minimal model of D. Let x be the least point in M with the property that $\Phi[t] \subseteq M[x]$ for some minimal vertex t of Φ. Write the model as the flexi-word $M = M_1 < M[x] < M_2$, where M_1 and M_2 are flexi-words and the symbol $M[x]$ in this expression corresponds to the point x.

Define T to be the set of all variables v such that (i) v is minor in the graph of Φ and (ii) for every path $u_0, u_1, \ldots, u_n = v$ in the graph of Φ which passes only through edges labelled '$\leq$', we have $\Phi[u_i] \subseteq M[x]$ for all $i = 0 \ldots n$. Let $\Phi \setminus T$ be the query whose graph corresponds to the graph obtained by deleting the vertices T from the graph of Φ.

We claim that $M_2 \models p$ for all $p \in Paths(\Phi \setminus T)$. For, suppose $p \in Paths(\Phi \setminus T)$ and let t be the minimal vertex in the graph of p. Clearly $t \notin T$. There are two cases.

I. Part (i) of the definition of T fails, i.e., t is not minor. In this case there exists a path of the form $q = a_1 r_1 a_2 r_2 \ldots r_{n-1} a_n r_n p$ in Φ, where for some $i = 1 \ldots n$ we have $r_i =$ '$<$'. We have assumed M satisfies all paths of Φ, Hence $M \models q$. But x is the least point at which $a_1 \subseteq M[x]$. It follows that $M_2 \models p$.

II. Part (ii) of the definition of T fails. Then there exists a path $q = a_1 \leq a_2 \leq \ldots \leq a_n \leq p$ in Φ, where either $\Phi[t] \not\subseteq M[x]$ or for some $i = 1 \ldots n$ $a_i \not\subseteq M[x]$. Again $M \models q$, and it follows from the fact that x is the least point such that $a_1 \subseteq M[x]$ that $M_2 \models p$.

This completes the proof that $M_2 \models p$ for all $p \in Paths(\Phi \setminus T)$. It follows by the induction hypothesis that $M_2 \models \Phi \setminus T$. Let θ' be a satisfying assignment for $\Phi \setminus T$ in M_2, and extend this to an assignment for Φ by defining $\theta(v) = x$ for all $v \in T$ and $\theta(v) = \theta'(v)$ otherwise. We claim that θ is a satisfying assignment for Φ in M. This will complete the proof.

First, note that for all $v \in T$, we have $\Phi[v] \subseteq M[x]$, by definition of T. Hence all proper atoms of Φ are satisfied under the assignment, and it suffices to consider the order atoms. If $u < v$ is an atom in Φ, then we cannot have $v \in T$. If both u and v are not in T then the atom is in $\Phi \setminus T$, hence obviously satisfied. If $u \in T$ then $\theta(u) = x$, and since $\theta(v)$ lies in M_2 it is greater that x, hence the atom $u < v$ is satisfied. Finally, if $u \leq v$ is an atom in Φ and $v \in T$

then we must have $u \in T$ also. Since $\theta(u) = \theta(v) = x$, the atom is satisfied. On the other hand, if $v \notin T$ then we argue exactly as before. □

Lemma 4.1 shows that the problem of answering arbitrary conjunctive queries may be reduced to the problem of answering sequential queries. The next result shows that to answer sequential queries it suffices to restrict attention to databases of width one.

Lemma 4.2: If p is a sequential query then $D \models p$ if and only if $q \models p$ for some path q of the database D.

Proof: It is clear that if $q \models p$ for some path q of the database D then D entails p. For the converse, we show by induction on $|D| + |p|$ that if $q \models p$ for no path q of D then $D \not\models p$. The base case, when D and p are both empty, is trivial. Suppose that the claim holds for all pairs of database and query of combined size less than $|D| + |p|$. Assume that $q \models p$ for no path q of D. We consider three cases.

Case I: *The flexi-word corresponding to p has the form $a\sigma$ and there exists a minimal vertex u of the graph of D such that $a \not\subseteq D[u]$.* Here a is a subset of *Pred* and σ is a sequence of symbols, possibly null. Let $D' = D \setminus \{u\}$. There is no path q of D such that $q \models p$, so there can be no path q of D' such that $q \models p$. Hence, since $|D'| + |p| < |D| + |p|$, it follows by the induction hypothesis that $D' \not\models \Phi$. That is, there exists a flexi-word M representing a model of D' such that $M \not\models p$. Now the flexi-word $D[u] < M$ represents a model of D. If it were the case that $D[u] < M \models p$, then we would have $M \models p$, because $a \not\subseteq D[u]$. Thus $D[u] < M \not\models p$, from which it follows that $D \not\models p$.

Case II: *The flexi-word corresponding to p is of the form $a < p'$ and for all minimal vertices u of the graph of D we have $a \subseteq D[u]$.* Here a is a subset of *Pred*, and p' is a flexi-word. In this case, let S be the set of minor vertices of the graph of D, and put $D' = D \setminus S$. Suppose that there exists a path q' in D' such that $q' \models p'$. Let v be the vertex of the graph of D corresponding to the minimal vertex of this path. Since v is not in S, hence not minor in D, there exists a path in the graph of D which corresponds to a flexi-word $q = a_1 r_1 a_2 r_2 \ldots a_n r_n q'$, where at least one of the r_i is '$<$'. Since $a \subseteq a_1$, it follows that $q \models p$, a contradiction. This shows that $q' \models p'$ for no path q' of D'. By the induction hypothesis, we obtain that there exists a flexi-word M' corresponding to a model of D' such that $M' \not\models p'$. Let a' be the union of the sets $D[v]$ for $v \in S$, and let M be the flexi-word $a' < M'$. Clearly M corresponds to a model of D. However, $M \models a < p'$ if and only if $M' \models p'$, which is false. Hence $M \not\models p$, which proves that $D \not\models p$.

Case III: *The flexi-word corresponding to p is of the form $a \leq p'$ and for all minimal vertices u of the graph of D we have $a \subseteq D[u]$.* In this case, there exists no path q of D such that $q \models p'$. For, if this were the case, then the minimal vertex v of q would be minimal in D, hence satisfy $a \subseteq D[u]$. But this implies $q \models a \leq p'$, a contradiction. It follows by the induction hypothesis that there exists a model M of D such that $M \not\models p'$. A fortiori, $M \not\models a \leq p'$, which shows that $D \not\models p$. □

```
$SEQ(D,p)$:

IF $p$ is empty THEN return $true$;
IF $D$ is empty THEN return $false$;
IF $p = a\sigma$ and there exists a minimal vertex $u$ of
   the graph of $D$ with $a \not\subseteq D[u]$
   THEN return $SEQ(D \setminus \{u\}, p)$;
IF $p = a < p'$ THEN return $SEQ(D \setminus \{v|v \text{ is minor in } D\}, p')$;
IF $p = a \leq p'$ THEN return $SEQ(D, p')$.
```

Figure 6: An algorithm for sequential queries

The proof above yields a recursive procedure, shown in Figure 6, which given as input a database D and a sequential query p decides if $D \models p$. (In fact, we can also modify the algorithm so that it returns a model of D in which p fails, if $D \not\models p$.) If $Pred$ is the set of predicate symbols, then the procedure can be implemented to run in time $O(|D| \cdot |p| \cdot |Pred|)$.

To see this, let us first recall that simply performing a topological sort (in the standard sense of the term) of a dag can be done in linear time. The main difficulty here is to be able to efficiently calculate after each vertex deletion the new set of minimal vertices. The standard trick for this (see for example [14]) is to associate with each vertex v the number of vertices u such that there exists a edge from u to v. Whenever we delete a vertex, we decrement the count of all of its successors. Should this operation decrease the count associated with a vertex to zero, this vertex is added to the list of current minimal elements. This results in a linear time topological sort.

Another operation we need to be able to perform efficiently is the deletion of the set of minor vertices. To do this, we simply start deleting minimal vertices one at a time, but each time one of these vertices has a successor through an edge labelled '$<$', we mark this successor. We take care to delete unmarked minimal vertices only; once all minimal vertices are marked, we are done. It is straightforward to show that this procedure is correct and has no more than linear total cost over the run of the algorithm.

The main contribution to complexity, then, is the determination of whether $p[u] \subseteq D[v]$ for all minimal vertices v of D, where u is a variable of p. Once these sets have been sorted, at cost $O((|D|+|p|)|Pred| log\,|Pred|) = O(|D|\cdot|p|\cdot|Pred|)$ the containment can be tested in time $2k$. In the worst case we need to perform the containment test $|D| \cdot |p|$ times, with total cost $O(|D| \cdot |p| \cdot |Pred|)$. (The worst case occurs when all order relations in p are '$\leq$', D contains no order atoms, and $p[u] \subseteq D[v]$ for all u and v.) This establishes the bound claimed above. We note that if p does not contain the relation '$\leq$' then the maximum number of set comparisons we need to perform is $|D| + |p|$, so in this case the total cost of the comparisons decreases to $O((|D| + |p|)|Pred|)$.

Corollary 4.3: The combined complexity for arbitrary monadic databases and sequential queries is in polynomial time.

By Lemma 4.1, a database entails a conjunctive query Φ just in case it entails each path of Φ. If the query Φ is fixed, so is its set of paths, and we obtain the following upper bound on data complexity.

Corollary 4.4: The data complexity of conjunctive monadic queries with respect to monadic databases is in linear time.

Note, however, that the number of paths of a query can grow exponentially in the size of the query, so the constant of proportionality may be very large. It also follows from this that an algorithm which tests entailment of each path of a query will not run in polynomial time if the size of the query is permitted to grow. This suggests that the combined complexity when queries are non-sequential may be high. As we will show shortly, this is indeed the case.

In case p and q are flexi-words in which all order relations are '$<$', we may obtain a particularly simple characterization of when $q \models p$. Let us call such flexi-words simply *words*, and write them as sequences of symbols over the alphabet $A = \mathcal{P}(Pred)$. That is, we write $a_1 a_2 \ldots a_n$ for the word $a_1 < a_2 < \ldots < a_n$. If $p = a_1 \ldots a_n$ and $q = b_1 \ldots b_m$ are words in A^* then we will say that p is a *subword* of q if there exist indices $i_1 {<} i_2 {<} \ldots {<} i_n$ such that for each $j = 1 \ldots n$, the set a_j is a subset of the set b_{i_j}. For example, the word $\{P,Q\}\{P\}\{R\}$ is a subword of the word $\{P,Q,R\}\{R\}\{P,R\}\{P,Q,R\}$. (If an element of a word is a singleton set then we will omit the braces. Thus, the first of the two flexi-words above will also be written as $\{P,Q\}PR$.) We obtain from the proof of Lemma 4.2 the following.

Proposition 4.5: If p and q are words then $q \models p$ if and only if p is a subword of q.

By Lemma 4.1 a database entails a conjunctive query Φ just in case it entails every path of Φ. Combining this with Lemma 4.2, we obtain the following characterization of entailment of conjunctive queries: D entails Φ just in case for every path p of Φ there exists a path q of D such that $q \models p$. In case D and Φ contain only the order relation '$<$', this holds just when every path of Φ is a subword of some path of D. The proof of the following lower bound makes use of this characterization.

Theorem 4.6: The combined complexity of $\{<\}$-databases and width two conjunctive $\{<\}$-queries over a fixed set of two monadic predicates is co-NP hard.

Proof: We use a reduction from the problem of determining if a formula in disjunctive normal form is a tautology. Since this is the complement of determining if a conjunctive normal form formula is satisfiable, this is a co-NP hard problem. Suppose that $\alpha = \bigvee \delta_i$ is a disjunctive normal form formula over the propositional constants $P_1, \ldots, P_m$ where the δ_i are conjunctions of literals.

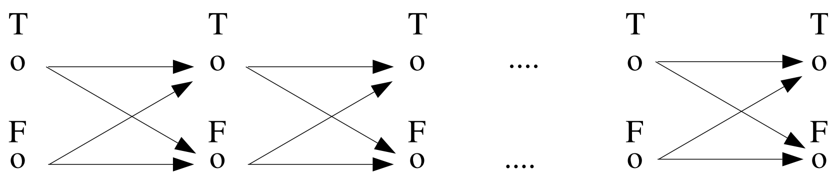


Figure 7: The query $\Phi(\alpha)$.

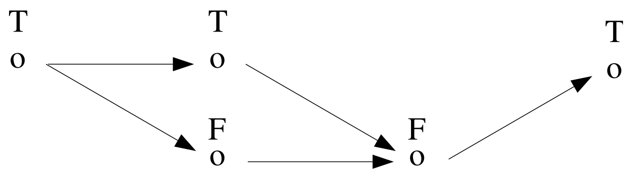


Figure 8: The graph of the conjunction $P_1 \wedge \overline{P_3} \wedge P_4$.

We fix the set $\{T, F\}$ of predicates. All paths in the database or the query will be words of length m over the alphabet $\{T, F\}$, and correspond to valuations of the set of propositional constants. Define $\Phi(\alpha)$ to be the query whose dag is depicted in Figure 7. Here there are two rows, each containing m vertices. Thus the query $\Phi(\alpha)$ has width two. Clearly $\mathit{Paths}(\Phi) = \{T, F\}^m$ is the set of all words of length m over $\{T, F\}$.

The database $D(\alpha)$ corresponding to the formula α will be the union of a number of disconnected components, one for each disjunct δ_i. Each component will be isomorphic to the subgraph of the graph of $\Phi(\alpha)$ generated by set of nodes chosen as follows. For each $j = 1 \ldots m$, if the disjunct contains neither the literal P_j nor the literal $\overline{P_j}$ then we retain both vertices in the j-th column of $\Phi(\alpha)$. If the disjunct contains the literal P_j then we retain from the j-th column of $\Phi(\alpha)$ only the vertex labelled T. Finally, if the disjunct contains the literal $\overline{P_j}$ then we retain from the j-th column of $\Phi(\alpha)$ only the vertex labelled F. (We assume that each disjunct is consistent.) For example, the graph of the component corresponding to the conjunction $P_1 \wedge \overline{P_3} \wedge P_4$ is illustrated in Figure 8. We let $D(\alpha)$ be the disjoint union of the components corresponding to the disjuncts δ_i. Note that all paths of $D(\alpha)$ have length m. It is readily seen that a word is a path of $D(\alpha)$ just in case α is true under the corresponding valuation of the propositional constants.

Since the paths of $D(\alpha)$ and $\Phi(\alpha)$ all have length m, a path of $\Phi(\alpha)$ is a subword of a path of $D(\alpha)$ just in case it is in fact a path of $D(\alpha)$. Thus $D(\alpha)$ entails $\Phi(\alpha)$ if and only if every word in $\{T, F\}^m$ is a path of $D(\alpha)$. Interpreted in terms of valuations, this holds exactly when α is true under every valuation, that is, when α is a tautology. $\Box$

We will see shortly, (Proposition 5.2), that this lower bound is the best possible, and that even disjunctive monadic queries have combined complexity in co-NP. We note that the lower bound may also be established using only $\{\leq\}$-databases and $\{\leq\}$-queries. To do so, we introduce two new predicates, P

and Q. The proof then proceeds using databases and queries whose dags have the same structure as those in the proof of Theorem 4.6, except that the edges are labelled '$\leq$' instead. The vertices are labelled as before, but in addition we label the vertices in the odd numbered columns with P and those in even numbered columns with Q, so that we obtain paths of the form

$$p = \{P, R_1\} \leq \{Q, R_2\} \leq \{P, R_3\} \ldots \leq \{U, R_n\}$$

where the R_i are either T or F, and U is P or Q according as n is odd or even. One may verify using the algorithm of Figure 6 that if

$$q = \{P, R_1'\} \leq \{Q, R_2'\} \leq \{P, R_3'\} \ldots \leq \{U, R_n'\}$$

is a flexi-word of the same form and length, then $q \models p$ if and only if $q = p$. It follows from this that the proof works as before.

Notice that the databases $D(\alpha)$ constructed in the proof of Theorem 4.6 may grow to have arbitrary width, since the formula α may have an arbitrary number of disjuncts. In such applications as databases recording the reports of a fixed number of observers, it is natural to assume a width bound. Hence, it is of interest to determine whether constraining the database to have width bounded by a constant results in a decrease in complexity. The following result shows that it does.

Theorem 4.7: If D is a database of width k and Φ is a conjunctive monadic query then $D \models \Phi$ can be decided in time $O(|D|^{k+1} \cdot |\Phi|)$.

Proof: We reduce the problem to a depth first search of a directed graph which represents the possible calling sequences of the algorithm SEQ when given as input the database D and a path of the query Φ.

If S is a set of vertices of the graph of D we will write $D \uparrow S$ for the database corresponding to the subgraph of the graph of D consisting of the vertices S and all vertices reachable from these. Clearly if S is an antichain then S is the set of minimal vertices of $D \uparrow S$. If D is a database and S is a set of vertices of its graph we write $D \setminus S$ for the database whose graph is obtained from the graph of D by deleting the vertices S.

Define $\mathcal{T}$ be the set of tuples of the form (S, u) where S is an antichain of the graph of D and u is a vertex of the graph of Φ. Say that the tuple (S, u) is *initial* if S is the set of minimal vertices of the graph of D and u is a minimal vertex of the graph of Φ. We construct a directed graph with vertices $\mathcal{T}$. There exists an edge from (S_1, u) to (S_2, v) just in case one of the following holds.

(a) There exists $s \in S_1$ such that $\Phi[u] \not\subseteq D[s]$, S_2 is the set of minimal vertices of the graph of $(D \uparrow S_1) \setminus \{s\}$, and $u = v$. (Remark: it suffices to have at most one edge of this type from each tuple (S_1, u).)

(b) There does not exists $s \in S_1$ such that $\Phi[u] \not\subseteq D[s]$, there exists in the graph of Φ an edge from u to v labelled '$<$', and S_2 is the set of minimal vertices of the graph of $(D \uparrow S_1) \setminus \{w | w \text{ is minor in } D \uparrow S_1\}$.

(c) There does not exists $s \in S_1$ such that $\Phi[u] \not\subseteq D[s]$, there exists in the graph of Φ an edge from u to v labelled '$\leq$', and $S_1 = S_2$.

It is not difficult to see that a tuple (S, v) will be reachable from some initial vertex (T, u) just when there exists a path $u_1, u_2, \ldots, u_n$ in the graph of Φ corresponding to the flexi-word $p = a_1 r_1 a_2 r_2 \ldots r_{n-1} a_n$ such that $u = u_1$, $v = u_j$ for some j, and the algorithm SEQ, given as input the database D and the flexi-word p makes the call $SEQ(D \uparrow S, a_j r_j a_{j+1} r_{j+1} \ldots r_{n-1} a_n)$ at some stage of the computation.

In particular, if a tuple of the form $(\emptyset, v)$ is reachable from some initial vertex, then the call $SEQ(D, p)$ eventually returns $false$, so the path p is not a consequence of the database, and $D \not\models \Phi$. Conversely, if no tuple of the form $(\emptyset, v)$ is reachable from any initial vertex, then there does not exist a path p of Φ such that $SEQ(D, p)$ returns $false$, hence $D \models \Phi$.

To determine the complexity bound, note that we may decide reachability of a vertex of the form $(\emptyset, v)$ by means of a depth first search. The most expensive nodes to process during this search are nodes (S_1, u) from which there exist edges of type (b). Here we must first determine that $\Phi[u] \not\subseteq D[s]$ for each $s \in S_1$. Since S_1 may contain at most the fixed number k elements, this may be done in time $O(|Pred|)$, once the vertex labels have been sorted. Next, we must determine the set S_2 of vertices which are minimal after the minor vertices of $D \uparrow S_1$ have been deleted. Using the ideas for the implementation of SEQ, this may be done in time linear in the number of vertices and edges deleted, certainly in time $|D|$. Finally, there is the cost associated with setting up and returning from the recursive call of the search procedure, for each edge from u labelled '$\leq$' in the graph of Φ. Whereas SEQ could destructively update the counters on the vertices of D used to calculate minimal vertices, the present depth first search must restore the values upon backtracking. The cost of this is at most $O(|D|)$, for each edge from (S_1, u) to be traversed. Thus the total cost of processing vertices (S_1, u) generating edges of type (b) is $O(\#succ(u) \cdot |D|)$, where $\#succ(u)$ is the number of successors of u in Φ. Summing over all $|D|^k \cdot |\Phi|$ vertices in $\mathcal{T}$, we find a total cost for the depth first search of $O(|D|^{k+1} \cdot |\Phi|)$. $\square$

Notice that when we consider the effect of applying the algorithm of Theorem 4.7 on fixed conjunctive queries, we obtain the bound $O(|D|^{k+1})$. Comparing this with the linear time algorithm of Corollary 4.4 we see that the present approach yields a less efficient compiled version of a fixed query. However, since the former algorithm has a constant of proportionality of order $2^{|\Phi|}$, it is not immediately clear which algorithm will be more efficient in practice.

5 Disjunctive Monadic Queries

The previous section dealt with the complexity of conjunctive monadic queries. In this section we consider expression complexity and combined complexity of monadic queries in the disjunctive case. In doing so, we present an algorithm for disjunctive monadic queries that is able to efficiently generate all countermodels in case the query is not entailed by the data. The next section will deal with data complexity in the disjunctive case.

We begin by noting that the algorithm for conjunctive queries of Theorem 4.7 yields a more efficient way to answer monadic queries in a given model than the naive approach of considering all substitutions for the variables of the query. If M is a finite model and D_M is the database of width one with the same word representation as M, then M is the unique minimal model of D_M. Hence $M \models \Phi$ if and only if $D_M \models \Phi$. This yields the following corollary of Theorem 4.7.

Corollary 5.1: If M is a finite model and Φ is a disjunctive monadic query over the predicates $Pred$, then $M \models \Phi$ is decidable in time $O(|M| \cdot |\Phi| \cdot |Pred|)$. In particular, monadic databases have linear time expression complexity with respect to disjunctive monadic queries.

Proof: We analyze the algorithm of Theorem 4.7 in the case that D is a width one database not containing the relation '$\leq$'. In this case the computation of minimal vertices can be done in constant time and does not require book-keeping, so the cost of processing each node of $\mathcal{T}$ reduces to the $O(|Pred|)$ containment test together with the $O(\#succ(u))$ cost for the edges traversed. Summing as before, we obtain the bound $O(|M| \cdot |\Phi| \cdot |Pred|)$ for conjunctive queries. This also applies to disjunctive queries, since a model satisfies a disjunctive query just when it satisfies some disjunct. □

Since databases containing binary predicates have NP complete expression complexity, we note a decrease in complexity for monadic predicates. As a further corollary we obtain an upper bound on the combined complexity of monadic disjunctive queries, which improves the upper bound found in the binary case. To determine $D \models \Phi$ it suffices to verify Φ in each of the minimal models of D. By Corollary 5.1, this can be done in polynomial time. Thus, we have

Proposition 5.2: The combined complexity of monadic databases and disjunctive queries is in co-NP.

This result indicates an improvement upon the Π_2^p completeness of combined complexity for queries containing binary predicates. Theorem 4.6 shows that the upper bound is optimal.

In the case of monadic conjunctive queries, we discussed two distinct restrictions, query sequentiality and bounded width of the database, under which combined complexity drops from co-NP complete to PTIME. We now seek a more refined understanding of the influence of these parameters on the complexity of disjunctive queries. The following result states an upper bound in terms of the database width and structure of the query.

Theorem 5.3: If D is a database of width k and Φ is the query $\Phi_1 \vee \ldots \vee \Phi_n$, where each Φ_i is a conjunctive monadic query using predicates $Pred$, then $D \models \Phi$ can be decided in time $O(|D|^{2k} \cdot |Pred| \cdot \Pi_{i=1\dots n}|\Phi_i|)$. In particular, the combined complexity of monadic queries with a bounded number of disjuncts and databases of bounded width is in polynomial time.

Proof: As in the proof of Theorem 4.7, we reduce the problem to a depth first search of a directed graph. In this case, the vertices of this graph consist of a number of components. One of these components corresponds to a class of partial topological sorts of the database. Each remaining component corresponds to a partially evaluated set of paths of one of the disjuncts of the query.

Define $\mathcal{T}$ be the set of tuples of the form $(S, T, u_1, \dots, u_n, x_1, \dots, x_n)$, where

1. S and T are antichains of the graph of D,
2. for each i, u_i is a vertex of the graph of Φ_i, and
3. each x_i is either 0 or 1.

We will write $a(S, T)$ for the union of the sets $D[u]$ where u is a vertex of the graph $D(S, T) = (D \uparrow S) \setminus (D \uparrow T)$, which consists of the vertices of D which lie after S, but strictly before T. Intuitively, a tuple represents a situation in which the database D has been partially topologically sorted, with $D \uparrow (S \cup T)$ being the unsorted portion, and $D(S, T)$ provisionally the set of minor vertices to be mapped to the next point of the model being constructed. (Other minor vertices may be added later.) Each u_i represents that some path of the query Φ_i is satisfied up to, but not yet including, the vertex u_i. The meaning of the x_i is explained below.

Say that the tuple $(S, T, u_1, \dots, u_n, x_1, \dots, x_n)$ is *initial* if $S = \emptyset$, T is the set of minimal vertices of the graph of D and for each i, $x_i = 0$ and u_i is a minimal vertex of the graph of Φ_i. A tuple is *final* if T is the empty set. We construct a directed graph with vertices $\mathcal{T}$. There exists an edge from $(S, T, u_1, \dots, u_n, x_1, \dots, x_n)$ to $(S', T', u'_1, \dots, u'_n, x'_1, \dots, x'_n)$ just in case one of the following holds.

(a) There exists $v \in T$ such that v is minor in $D \uparrow (S \cup T)$, and we have that S' is the set of minimal vertices of $D \uparrow (S \cup \{v\})$, T' is the set of minimal vertices of $(D \uparrow T) \setminus \{v\}$, and for each i, $x'_i = x_i$ and $u'_i = u_i$. (This type of edge corresponds to adding the vertex v to the set of vertices to be mapped to the next point of the linear order.)

(b) We have that (i) $j \in \{1 \dots n\}$ is the least number such that $x_j = 0$ and $\Phi_j[u_j] \subseteq a(S, T)$ and (ii) there exists an edge from u_j to u'_j in Φ_j. In this case $S' = S$, $T' = T$, and for each $i \neq j$ we have $u'_i = u_i$ and $x'_i = x_i$. If the edge from u_j to u'_j is labelled '$<$' then $x'_j = 1$, else $x'_j = 0$. (This type of edge corresponds to noting that any path of Φ_j which has already been satisfied up to (but not including) the vertex u_j, can be satisfied up to the vertex u'_j by interpreting this vertex in the point of the linear order being constructed. If the edge from u_j to u'_j is labelled '$<$' then u'_j cannot also be satisfied by means of this point. The variable x_j keeps track of when this is the case.)

(c) For each i, either $\Phi_i[u_i] \not\subseteq a(S, T)$ or $x_i = 1$, and we have $S' = \emptyset$, $T' = T$ and for each i, $u'_i = u_i$ and $x'_i = 0$. (This type of edge corresponds to mapping the vertices $D(S, T)$ to the next point of the linear order, and proceeding to construction of the following point. Note that we do not do so until there is no possibility of extending one of the successful paths to include u_i, given the current choice $D(S, T)$ of minor vertices.)

Using the interpretations of this construction given above, it may be shown that a final tuple $(S, T, u_1, \ldots, u_n, x_1, \ldots, x_n)$ is reachable from some initial tuple just when there exists a minimal model M of D, such that for each $i = 1 \ldots n$ there exists a path $v_1, v_2, \ldots, v_m = u_i$ in the graph of Φ_i, corresponding to the flexi-word $p = a_1 r_1 a_2 r_2 \ldots r_{m-2} a_{m-1} r_{m-1} a_m$, such that M satisfies the flexi-word $a_1 r_1 a_2 r_2 \ldots r_{m-2} a_{m-1}$, but not the flexi-word p. It follows from this that a final tuple is reachable just when $D \not\models \Phi$.

In order to implement the depth first search, we may use again some of the ideas for the algorithm SEQ of the previous section. Note that we may have up to k edges of type (a) from any tuple, at most one edge of type (b), and at most one edge of type (c). Besides the time taken to traverse these edges, the only other cost associated with a tuple is the determination of $\Phi_i[u_i] \subseteq a(S, T)$ for $i = 1 \ldots n$. As in the algorithm of Theorem 4.7, when backtracking we need to restore the values of the counters used to compute minimal vertices, as well as the markers used to decide whether a vertex is minor. This may be done in constant time, since we are deleting one vertex of D at a time, hence affecting at most $2k$ successors. Hence each tuple may be processed in time $O(k + n \cdot |Pred|)$. Since the antichains have size at most k, the number of tuples is $O(|D|^{2k} \cdot \Pi_{i=1 \ldots n} |\Phi_i|)$, and the total cost for the depth first search is $O(|D|^{2k} \cdot |Pred| \cdot \Pi_{i=1 \ldots n} |\Phi_i|)$. $\Box$

We note that for conjunctive queries, the PTIME algorithm of Theorem 4.7 is more efficient than the present algorithm, since the dependence on database size is $O(|D|^{k+1})$ instead of $O(|D|^{2k})$. We do not know if this discrepancy can be remedied. However, we point out that the present algorithm computes more information than the former, so it is likely to remain of interest even if the bound can be improved. Suppose we have a database D which does not entail the query Φ, and we are interested in the models of D in which Φ fails. (For example, the database may represent a set of scheduling constraints, and the negation of the query the integrity constraints to be satisfied.) The algorithm may be modified to enumerate all such models in an efficient manner.

Each path in the graph of $\mathcal{T}$ from an initial tuple to a final tuple corresponds to topological sort of the database producing a model in which the query is false. Conversely, each such model corresponds to some path (possibly many) of the graph of $\mathcal{T}$. The algorithm may be readily modified to prune vertices of $\mathcal{T}$ from which no final vertex is accessible. This results in a graph whose maximal paths correspond to models in which the query fails. By traversing this graph, we enumerate (with some redundancy) all the models of interest, with no more than polynomial time delay between successive outputs.[3]

The bound of Theorem 5.3 indicates an exponential dependence on the width of the database and upon the number of disjuncts. We already know from Theorem 4.6 that (for queries of width two or more) the exponential dependence on the width of the database is unlikely to be eliminable. The following results show that this is also the case for the exponential dependence on the number of disjuncts.

Proposition 5.4: The combined complexity of bounded $\{<\}$-databases over four monadic predicates and $\{<\}$-queries with an unbounded number of sequential disjuncts is co-NP hard.

Note that this result indicates that the exponential dependence on the number of disjuncts will hold even if we make the disjuncts sequential, a constraint we showed to lead to PTIME combined complexity for conjunctive queries (Corollary 4.3). Moreover, it can be shown that sequentiality also does not help to relax the width constraint of the PTIME class of Theorem 5.3 when we bound the number of disjuncts:

Proposition 5.5: The combined complexity of bounded disjunctions of sequential $\{<\}$-queries and unbounded width $\{<\}$-databases predicates is co-NP hard.

This indicates that the PTIME class of Theorem 5.3 is the unique maximal class with respect to the parameters we have considered in this paper for combined complexity of monadic queries. For the proofs of Proposition 5.4 and Proposition 5.5 we refer the reader to [23].

6 Data Complexity of Disjunctive Monadic Queries

We have seen that the data complexity of conjunctive monadic queries is in polynomial time. The data complexity of disjunctive monadic queries also turns out to be in polynomial time, but we will need some more sophisticated techniques to show this. We will be able to prove that each query has data complexity in PTIME, but without being able to provide an explicit algorithm that works with this complexity. The proof relies on the following concept.

Definition 6.1: A *quasi-order* on a set X is a reflexive transitive relation $\preceq$. An element x of a quasi-ordered set X is minimal if for all $y \in X$,

[3]It seems this approach will work for a more expressive class of queries. It would be interesting to explore the relationships between this result and the dynamic programming techniques used to find optimal alignments of sequences [20].

$y \preceq x$ implies $x \preceq y$. The pair $(X, \preceq)$ is said to be a *well-quasi-order* if every nonempty subset S of X has a finite basis, i.e. a finite set $B \subseteq S$ such that $S = \{y \in S | x \preceq y\}$.

Equivalent definitions are as follows. First, a set is well-quasi-ordered just in case (a) every strictly decreasing sequence is finite and (b) every set of pairwise incomparable elements is finite. Alternately, call an infinite sequence $x_0, x_1, \ldots$ *bad* if there do not exist indices $i < j$ such that $x_i \preceq x_j$. Then X is well-quasi-ordered if and only if there does not exist a bad infinite sequence. The theory of well-quasi-orders is well developed: see Kruskal [19] for an historical survey and Milner [25] for a more detailed introduction. If X is a quasi-ordered set then we may define a quasi-order on the set $\mathcal{FP}(X)$ of finite subsets of X by $S_1 \preceq S_2$ when for each $x \in S_1$ there exists $y \in S_2$ with $x \preceq y$. Similarly, the set X^* of all finite sequences on X may be quasi-ordered by $x_1 \ldots x_n \preceq y_1 \ldots y_m$ if there exists a sequence of indices $i_1 < i_2 < \ldots < i_n$ with $x_j \preceq y_{i_j}$ for $j = 1 \ldots n$. It is known that if X is well-quasi-ordered then so are $\mathcal{FP}(X)$ and X^*. (If $X = \mathcal{P}(A)$ is the powerset of a finite set A, ordered by containment, the order on X^* is just the subword relation of Section 4. Thus, the fact that X^* is well-quasi-ordered means that the subword relation is a well-quasi-order.) We will also use the following lemma, whose proof is straightforward.

Lemma 6.2: Let X be a well-quasi-ordered set. Suppose Y is any set and let f be a function from Y to X. Then the relation $\sqsubseteq$ induced on Y by $x \sqsubseteq y$ when $f(x) \preceq f(y)$ is a well-quasi-order.

We now set about constructing a well-quasi-order on the set of monadic databases. We begin with an order on flexi-words. For flexi-words p, q define $p \preceq q$ if $q \models p$. The following result generalizes the fact that the subword relation is a well-quasi-order.

Lemma 6.3: The relation $\preceq$ well-quasi-orders the set of flexi-words over a finite set *Pred*.

Proof: We adapt a standard argument (cf. [25] Theorem 1.6) used to show well-quasi-orderedness of X^*. We show that there is no bad sequence. Suppose such a sequence $p_0, p_1, p_2, \ldots$ exists, where each flexi-word p_i is of the form $a_i r_i p'_i$. Here a_i is a subset of *Pred*, r_i is either '$<$' or '$\leq$' and p'_i is a flexi-word. We establish a contradiction. Without loss of generality, we may assume that for each i, p_i is a flexi-word of minimal length such that $p_0, p_1, \ldots, p_i$ may be extended to an infinite bad sequence. Since *Pred* is finite, there must exist an infinite sequence $i_0, i_1, \ldots$ of indices such that for all j, $a_{i_j} \subseteq a_{i_{j+1}}$ and the r_{i_j} are all the same order relation. Consider the sequence of flexi-words $p_0, p_1, \ldots, p_{i_0 - 1}, p'_{i_0}, p'_{i_1}, \ldots$. Since the original sequence was bad, there do not exist indices $i < j < i_0$ such that $p_i \preceq p_j$. On the other hand, if there were to exist indices $i < i_0$ and $j \geq 0$ such that $p_i \preceq p'_{i_j}$ then we would have $p'_{i_j} \models p_i$, hence $p_{i_j} = a_{i_j} r_{i_j} p'_{i_j} \models p_i$, i.e., $p_i \preceq p_{i_j}$, contradicting the assumption that the original sequence was bad. Thus, there must exists indices $j < j'$ such that $p'_{i_j} \preceq p'_{i_{j'}}$, i.e., $p'_{i_{j'}} \models p'_{i_j}$. Since $a_{i_j} \subseteq a_{i_{j'}}$ and $r_{i_j} = r_{i_{j'}}$, it follows that

$p_{i_{j'}} \models p_{i_j}$, i.e., $p_{i_j} \preceq p_{i_{j'}}$, again contradicting the assumption that the original sequence was bad. It follows that the new sequence is bad. But since p'_{i_0} has length shorter than p_{i_0}, and we assumed minimality of the original sequence, this is a contradiction. □

We now construct a well-quasi-order on the set $\mathcal{M}$ of monadic databases. Fix a finite set *Pred* of monadic predicates. (Since we are considering data complexity, only the predicates which occur in the query are relevant.) By Lemma 6.3 the set $FW(Pred)$ of flexi-words over *Pred* is well-quasi-ordered. It follows that the set $\mathcal{FP}(FW(Pred))$ is also well-quasi-ordered. Since for each database D we have $Paths(D) \in \mathcal{FP}(FW(Pred))$, by Lemma 6.2 we obtain a well-quasi-order on $\mathcal{M}$ defined by $D_1 \sqsubseteq D_2$ when $Paths(D_1) \preceq Paths(D_2)$.

Lemma 6.4: For any disjunctive monadic query Φ, if $D_1 \models \Phi$ and $D_1 \sqsubseteq D_2$ then $D_2 \models \Phi$.

Proof: If D is a database then the set of flexi-words $Paths(D)$ may also be interpreted as a database which contains a distinct linear sequence for each flexi-word in the set. Let ω be the function which maps finite models to their word representations by forgetting the interpretation of constants. Note that $p \in \omega(Mod_{\mathbf{Fin}}(D))$ if and only if $p \models \Psi_D$ where Ψ_D is the conjunctive query with the same graph as D. It follows by Lemma 4.1 that $\omega(Mod_{\mathbf{Fin}}(D)) = \omega(Mod_{\mathbf{Fin}}(Paths(D)))$ for any database D. Clearly if $Paths(D_1) \preceq Paths(D_2)$ then

$$\omega[Mod_{\mathbf{Fin}}(Paths(D_2))] \subseteq \omega[Mod_{\mathbf{Fin}}(Paths(D_1))]$$

so that $\omega[Mod_{\mathbf{Fin}}(D_2)] \subseteq \omega[Mod_{\mathbf{Fin}}(D_1)]$. Now notice that if $D_1 \models \Phi$ then $M \models \Phi$ for all $M \in Mod_{\mathbf{Fin}}(D_1)$, which holds just in case $p \models \Phi$ for all $p \in \omega[Mod_{\mathbf{Fin}}(D_1)]$. Clearly this implies that $p \models \Phi$ for all $p \in \omega[Mod_{\mathbf{Fin}}(D_2)]$, which implies that $D_2 \models \Phi$ also. □

Lemma 6.4 shows that for any fixed disjunctive query Φ, the set $S(\Phi)$ of monadic databases D satisfying $D \models \Phi$ is an ideal, that is, upwards closed. Thus to show that $D_2 \models \Phi$ it suffices to show that $D_1 \sqsubseteq D_2$ for some minimal element $D_1 \in S(\Phi)$. By the well-quasi-order property, these minimal elements exist and are finite in number. For any fixed D_1 we may determine $D_1 \sqsubseteq D_2$ in time linear in the size of D_2, as shown by Corollary 4.4. Thus we have

Theorem 6.5: The data complexity of monadic disjunctive queries on databases is in linear time.

Notice that this argument does not provide us with an explicit algorithm, since we do not as yet know how to calculate for each query Φ a finite basis of $S(\Phi)$, only that one exists. Thus, the proof is non-constructive.[4] Although we have established a linear time upper bound, this result is of little practical significance until an alternate constructive approach can be found, or further

[4]A number of other examples of non-constructive proofs that a set is in PTIME are known, see [8, 9].

analysis of the structure of the order $\sqsubseteq$ and the sets $S(\Phi)$ makes the present proof constructive. We also warn that the fact that combined complexity is co-NP hard indicates that the constants of proportionality may be very large.

One special case in which we do know how to compute a basis is when the query Φ is conjunctive. Lemma 4.1 and Lemma 4.2 together show that a conjunctive monadic query Φ is entailed by a database D just when $Paths(\Phi) \preceq Paths(D)$: this was the foundation of our earlier result that the data complexity of conjunctive monadic queries is in PTIME (Corollary 4.4). Thus, if we take D_Φ to be the database with the same labelled graph representation as the query Φ, then we see that $D \models \Phi$ if and only if $D_\Phi \sqsubseteq D$. This means that the set $S(\Phi)$ has unique minimal element D_Φ, which is straightforwardly computable. We see that not only does Theorem 6.5 subsume Corollary 4.4, but the proof we gave for the latter is also a special case of the proof of the former.[5]

7 Queries and Databases Containing Inequality

We have focussed for the bulk of the paper on the relations '$<$' and '$\leq$'. However, it is natural to generalize indefinite order databases by permitting databases and queries to contain atoms of the form $u \neq v$, where u and v are both order constants or variables. We now briefly comment on this generalization. Unfortunately, the consequences for our results are largely negative.

Let us first note that it is possible to eliminate occurrences of inequality by replacing each atom $u \neq v$ by the disjunction $u < v \vee v < u$. There are some cases in which this idea suffices to transfer PTIME results of this paper to more general situations. For example, this reduction shows that the data complexity of monadic $\{<, \leq, \neq\}$-queries in $\{<, \leq\}$-databases is still in linear time.

In another case it is is possible to get a partial generalization by a different method. Suppose we define the width of a $\{<, \leq, \neq\}$-database to be the width of the $\{<, \leq\}$-database obtained by deleting all the inequality atoms. Then, by appropriately modifying the procedure for topologically sorting a database, it is possible to generalize the proof of Theorem 5.3 to yield a $O(|D|^{2k}|\Phi|^l)$ algorithm for combined complexity of monadic $\{<, \leq\}$-queries Φ with l disjuncts in $\{<, \leq, \neq\}$-databases D of width k. Consequently, conjunctive monadic $\{<, \leq, \neq\}$-queries have PTIME data complexity in $\{<, \leq, \neq\}$-databases of bounded width.

In general, however, the reduction eliminating inequality clearly leads to an exponential blow-up in the size of databases and queries, so this approach is of little use. Indeed, as the following result shows, the apparent increase in complexity is real.

[5]We have also been able to obtain an algorithm for the basis computation under the assumption that all databases contain the relation '$<$' only. (Details to be reported elsewhere.) The general case, however, remains open.

Theorem 7.1:

1. There exists a $\{<\}$-database D of width one with NP hard expression complexity for conjunctive monadic $\{\neq\}$-queries.
2. There exists a sequential query with co-NP hard data complexity in monadic $\{\neq\}$-databases.

Proof: Both parts follow by reduction from graph three colourability.[6] Let G be a graph with vertices $V = \{v_1, \ldots, v_n\}$ and edges E. For the first part, take D to be the database $\{u_1 < u_2 < u_3, P(u_1), P(u_2), P(u_3)\}$ and let Φ be the query

$$\exists v_1 \ldots v_n[P(v_1) \wedge \ldots \wedge P(v_n) \wedge \bigwedge_{\{v_i,v_j\} \in E} (v_i \neq v_j)].$$

It is easy to show that $D \models \Phi$ if and only if G is three colourable.

For the second part, let Φ be the query

$$\exists t_1 t_2 t_3 t_4[P(t_1) \wedge P(t_2) \wedge P(t_3) \wedge P(t_4) \wedge t_1 < t_2 < t_3 < t_4]$$

and take $D = \{v_i \neq v_j | \{v_i, v_j\} \in E\} \cup \{P(v_i) | i = 1 \ldots n\}$. Then $D \models \Phi$ if and only if G is not three colourable. $\Box$

For conjunctive monadic queries we identified two cases with PTIME combined complexity: sequential queries and databases of bounded width. Theorem 7.1 suggests that (except for the special cases discussed above) neither case may be generalized to databases and queries containing the relation '$\neq$'. Furthermore, the PTIME data complexity of monadic queries is also likely to fail in databases containing '$\neq$' as a consequence of this result. There appears to be very little prospect of interesting tractable cases in the presence of inequality.

The proofs of Theorem 7.1 may be modified to use inequations only, suggesting that monadic $\{\neq\}$-databases and conjunctive $\{\neq\}$-queries have Π_2^p complete combined complexity, since the complexity of this problem is both NP hard and co-NP hard. While Π_2^p completeness in the case of databases containing binary predicates follows from results of [1], we do not know if monadic databases are subject to this bound.

Finally, we comment that the introduction of inequality requires a reconsideration of the influence of the order type on query evaluation. It is straightforward to check that the equivalence of the order types **Fin**, **Q** and **Z** with respect to entailment of tight queries continues to hold for $\{<, \leq, \neq\}$-queries. However, it is not clear that the reduction of the rational semantics to the finite semantics (Lemma 2.5) may be generalized. The difficulty here is that while it is possible to compute in polynomial time the set of order atoms entailed by a $\{<, \leq, \neq\}$-database (see Ullman [30] Section 14.2, van Beek and Cohen [3]), this set does not capture all the disjunctive consequences of the database. Koubarakis [18] has shown that it is possible to project a set of $\{<, \leq, \neq\}$-constraints onto a polynomial size representation which contains a number of disjunctions of inequations. However, we do not know whether a polynomial size disjunctive normal form representation is possible.

[6]Closely related results appear in [1, 32] for n-ary predicates and less restricted query forms.

8 Conclusion

Our results in this paper conform to a pattern noted elsewhere: reasoning about intervals is more complex than reasoning about points [13, 33]. We have seen that binary predicates, which in combination with the order relations permit the representation of certain forms of interval data, lead to increased complexity. Elsewhere, we show that the picture on interval data is actually not so bleak as it appears from the present paper [23, 22]. By replacing the bounded width constraint by a slightly more restrictive condition, called bounded concurrency, we obtain a tractable class of databases capable of expressing interval data. In fact, with this constraint, it is even possible to combine the order indefiniteness with certain forms of *recursive indefiniteness*, introduced in [24], while retaining tractability. The techniques used in this work generalize the ideas of Theorem 5.3.

Acknowledgements: This paper represents research done primarily at Rutgers University and completed during a visit to the University of New South Wales. Thanks to Thorne McCarty for comments which led to improvements in the presentation.

References

[1] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78:159–187, 1991.

[2] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:510–521, 1983.

[3] P. van Beek and R. Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6(3):132–144, 1990.

[4] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[5] A.K. Chandra and P.K. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 77–90, 1976.

[6] C.C. Chang and H.J. Keisler. *Model Theory.* North Holland, Amsterdam, 1973.

[7] V. Chvatal. *Linear Programming.* W.H. Freeman and Co., New York, 1983.

[8] M.R. Fellows and M.A. Langston. Nonconstructive advances in polynomial time complexity. *Information Processing Letters*, 26:157–162, 1987.

[9] M.R. Fellows and M.A. Langston. Nonconstructive tools for proving polynomial time decidability. *Journal of the ACM*, 35:727–739, 1988.

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

[11] M. C. Golumbic and R. Shamir. Complexity and algorithms for reasoning about time: A graph-theoretic approach. *Journal of the ACM*, 40(5):1108–1133, 1993.

[12] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[13] J. Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962, 1991.

[14] E. Horowitz and S. Sahni. *Data Structures in Pascal.* Computer Science Press, Rockville, MD, 1984.

[15] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. To appear, *Journal of Computer and System Sciences.* A preliminary version of this paper appears in *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on the Principles of Database Systems*, 1990, pp. 299-313.

[16] D.G. Kendall. Some methods and problems in statistical archeology. *World Archeology*, pages 68–76, 1969.

[17] A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, 35(1):146–160, 1988.

[18] M. Koubarakis. Dense time and temporal constraints with $\neq$. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 24–35, 1992.

[19] J.B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *Journal of Combinatorial Theory (Ser. A)*, 13:297–305, 1972.

[20] J.B. Kruskal. An overview of sequence comparison,: Time warps, string edits and macromolecules. *SIAM Review*, 25(2):201–327, 1983.

[21] J.L. Lassez. Querying constraints. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on the Principles of Database Systems*, pages 288–298, 1990.

[22] L.T. McCarty and R. van der Meyden. Reasoning about indefinite actions. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 59–70, 1992.

[23] R. van der Meyden. *The Complexity of Querying Indefinite Information: Defined Relations, Recursion and Linear Order.* PhD thesis, Rutgers University, 1992.

[24] R. van der Meyden. Recursively indefinite databases. *Theoretical Computer Science*, 116(1,2):151–194, 1993.

[25] E. C. Milner. Basic wqo- and bqo- theory. In I. Rival, editor, *Graphs and Orders*, pages 487–502. D. Reidel, 1985.

[26] R. Reiter. Towards a logical reconstruction of relational database theory. In M.L. Brodie, J. Mylopolous, and J.W. Schmidt, editors, *On Conceptual Modelling*, pages 163–189. Springer-Verlag, 1984.

[27] D.J. Rosenkrantz and H.B. Hunt. Processing conjunctive predicates and queries. In *Proceedings of the Sixth International Conference on Very Large Databases*, pages 64–72, 1980.

[28] E.D. Sacerdoti. *A Structure for Plans and Behaviour.* Elsevier, New York, 1977.

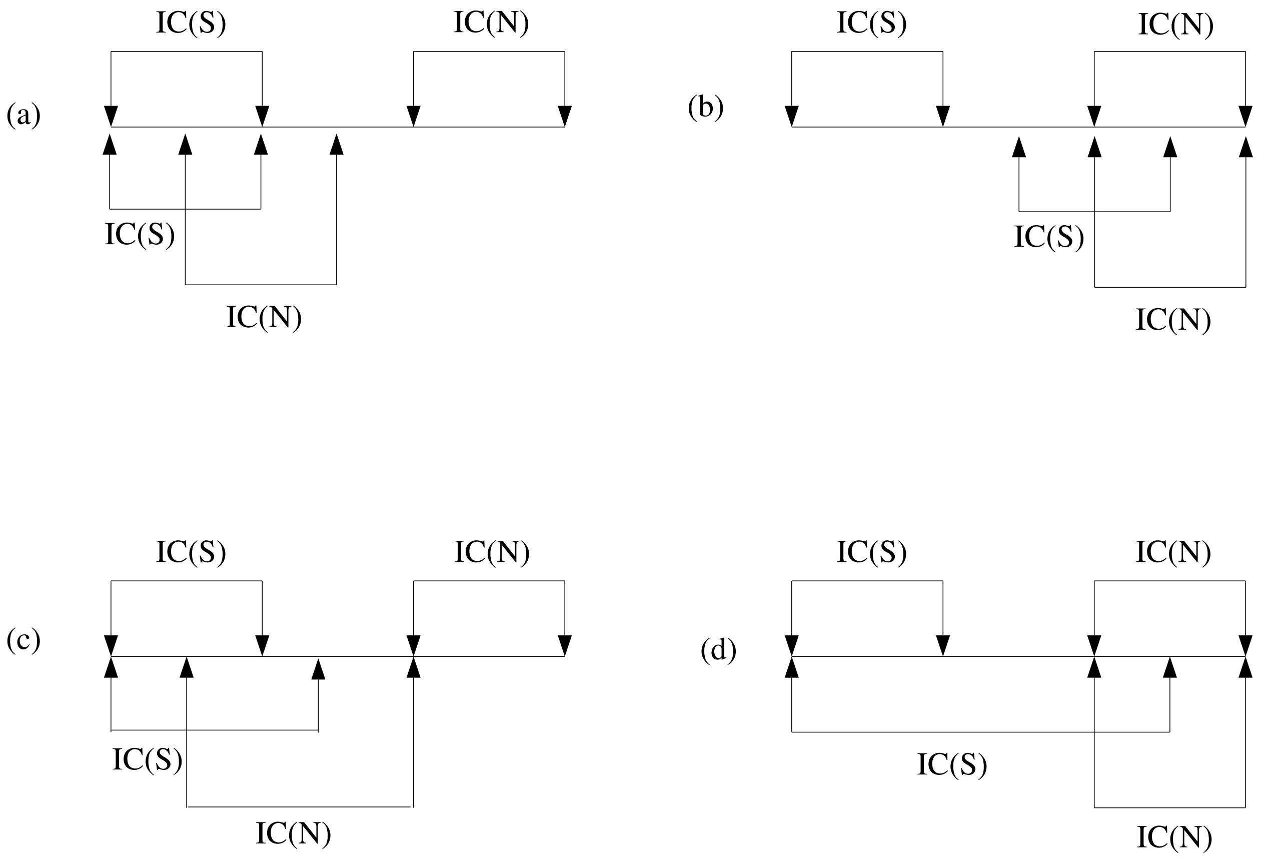
[29] D. Srivastava. Subsumption and indexing in constraint query languages with linear arithmetic constraints. *Annals of Mathematics and Artificial Intelligence*, 8(3-4), 1993.

[30] J.D. Ullman. *Principles of Database and Knowledge Base Systems, volume II: The New Technologies.* Computer Science Press, 1989.

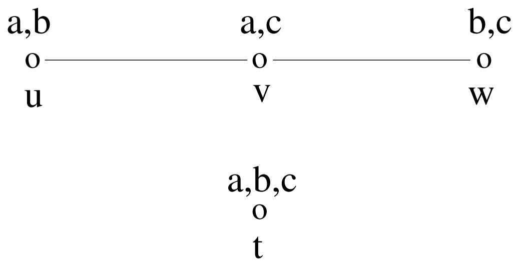
[31] M. Vardi. The complexity of relational query languages. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 137–146, 1982.

[32] M. Vardi. Querying logical databases. *Journal of Computer and System Sciences*, 33:142–160, 1986.

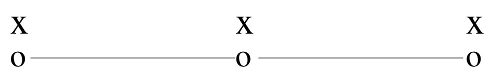
[33] M. Vilain, H. Kautz, and P. van Beek. Constraint propagation algorithms for temporal reasoning: a revised report. In Johan deKleer and Dan Weld, editors, *Readings in Qualitative Reasoning About Physical Systems.* Morgan Kaufmann, Los Altos, CA, 1989.

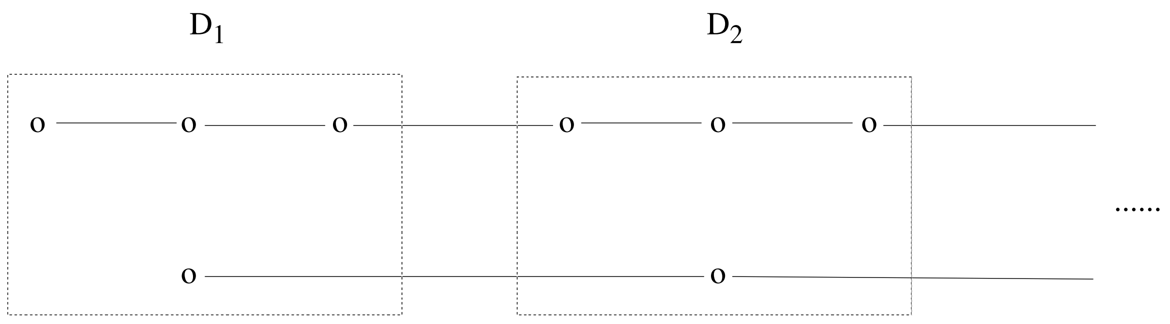

(a)
IC(S)
IC(N)
IC(S)
IC(N)
(b)
IC(S)
IC(N)
IC(S)
IC(N)
(c)
IC(S)
IC(N)
IC(S)
IC(N)
(d)
IC(S)
IC(N)
IC(S)
IC(N)

D:



φ(x):




D_1
D_2
......

$$\begin{tikzcd}
 & & \overset{S}{t_4} \\
\overset{P,Q}{t_1} \arrow[r] & \overset{P}{t_2} \arrow[ur] \arrow[dr, dashed] & \\
 & & \overset{R}{t_3}
\end{tikzcd}$$

$$\begin{tikzcd}
\overset{T}{o} \arrow[r] \arrow[dr] & \overset{T}{o} \arrow[r] \arrow[dr] & \overset{T}{o} & .... & \overset{T}{o} \arrow[r] \arrow[dr] & \overset{T}{o} \\
\overset{F}{o} \arrow[r] \arrow[ur] & \overset{F}{o} \arrow[r] \arrow[ur] & \overset{F}{o} & .... & \overset{F}{o} \arrow[r] \arrow[ur] & \overset{F}{o}
\end{tikzcd}$$

$$\begin{tikzcd}
\overset{T}{o} \arrow[r] \arrow[dr] & \overset{T}{o} \arrow[dr] & & \overset{T}{o} \\
& \overset{F}{o} \arrow[r] & \overset{F}{o} \arrow[ur] &
\end{tikzcd}$$