Available online at www.sciencedirect.com
SCIENCE DIRECT®

Artificial Intelligence 160 (2004) 173–190

Artificial Intelligence

www.elsevier.com/locate/artint

Research Note

An incremental algorithm for DLO quantifier elimination via constraint propagation

Matti Nykänen

Department of Computer Science, P.O. Box 68 (Gustaf Hällströmin katu 2B), FIN-00014 University of Helsinki, Finland

Received 19 February 2003; accepted 18 May 2004

Abstract

The first-order logical theory of dense linear order has long been known to admit quantifier elimination. This paper develops an explicit algorithm that yields an equivalent quantifier free form of its input formula. This algorithm performs existential quantifier elimination via constraint propagation. The result is computed incrementally using functional programming techniques. This approach may be of interest in implementing query languages for constraint databases.

Keywords: Quantifier elimination; Constraint propagation; Constraint databases

1. Introduction

Consider the language L of First Order Predicate Calculus (FOPC) with equality and the two-place infix relation symbol '$<$'. (We assume familiarity with the standard syntax and semantics of FOPC.) The *theory of dense linear order without endpoints* (*DLO*) consists of the following L-sentences [7, Eq. (2.32)].

$$\forall x.\neg x < x \tag{1}$$

$$\forall x, y.x < y \vee x = y \vee y < x \tag{2}$$

E-mail address: matti.nykanen@cs.helsinki.fi (M. Nykänen).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2004.05.011

$$\forall x, y, z.x < y \wedge y < z \rightarrow x < z \tag{3}$$

$$\forall x, y.x < y \rightarrow \exists z.x < z \wedge z < y \tag{4}$$

$$\forall x.\exists z.z < x \tag{5}$$

$$\forall x.\exists z.x < z. \tag{6}$$

The real line is a canonical model for this theory: The underlying set $\mathbb{R}$ of real numbers is strictly linearly ordered by magnitude (satisfying axioms (1)–(3)), this ordering is densely populated by these numbers (satisfying axiom (4)), and it extends indefinitely to both directions (satisfying axioms (5) and (6)). The rationals $\mathbb{Q}$ provide another model, since their natural order also satisfies these axioms.

DLO has long been known to admit *quantifier elimination* [7, Chapter 2.7]. That is, for every L-formula ϕ there exists an DLO-equivalent quantifier-free L-formula $\mathrm{QF}(\phi)$ in the following sense:

Definition 1. A formula ϕ is *DLO-valid*, in symbols $\models_{\mathrm{DLO}} \phi$, if and only if $\mathfrak{M} \models \phi\theta$ holds for every model $\mathfrak{M} = \langle \mathcal{U}_{\mathfrak{M}}, <_{\mathfrak{M}} \rangle$ of the DLO axioms (1)–(6) and assignment θ.

Two L-formulae ϕ and ψ are *DLO-equivalent*, in symbols $\phi \equiv_{\mathrm{DLO}} \psi$, if and only if $\models_{\mathrm{DLO}} \phi \leftrightarrow \psi$.

This paper presents an explicit algorithmic function QF. The algorithm reinterprets its argument ϕ as setting *constraints* on the possible orderings, and makes the quantifiers redundant by propagating these constraints.

However, the far more general problem of quantifier elimination over real closed fields which involves not only order but also addition and multiplication of real numbers has also long been known to be computable, with recent algorithmic advances [3]. Why then study this restricted subproblem?

The reason is that our algorithm is *incremental*, or able to produce its answers piece by piece. This property is in turn desirable in a database setting, as we shall discuss in Section 4 below. We hope our construction might be useful in attaining an incremental algorithm for the general problem. In addition, DLO quantifier elimination is also used in for example temporal inference [8], and therefore our algorithm might be of interest in such settings as well.

Consider the formula $\exists x_3.(x_1 < x_3) \wedge (x_3 < x_2)$ as a simple example of our approach. Eliminating this quantifier involves erasing every occurrence of the quantified variable x_3 from the formula. This erasure is in turn permitted only after we have made explicit all the implicit dependencies between the remaining variables x_1 and x_2 for which x_3 acts as an intermediary. This explication can be performed by inferring the implicit dependency $(x_1 < x_2)$ from the explicit dependencies $(x_1 < x_3)$ and $(x_3 < x_2)$. In other words, first we rewrite the original formula as $\exists x_3.(x_1 < x_3) \wedge (x_3 < x_2) \wedge (x_1 < x_2)$, and then we erase everything that involves x_3 from the rewritten formula to yield the answer $(x_1 < x_2)$. This paper shows that this approach suffices for quantifier elimination, except for a specific case which requires an additional heuristic.

The presentation proceeds as follows. First Section 2 presents the core idea of how constraints can be employed in quantifier elimination, provided that the quantified formula is

just an existentially quantified conjunction. Then Section 3 extends this idea to all formulae, and adds the necessary apparatus for incrementality. Finally Section 4 concludes the presentation and discusses the aforementioned database setting.

2. Quantifier elimination from conjunctions

In this section we concentrate on formulae of the kind

$$\exists x_n . \bigwedge_{1 \leqslant i < j \leqslant n} x_i R_{ij} x_j \tag{7}$$

or *conjunctions of atomic formulae enclosed within an existential quantifier* and show how the well-known Artificial Intelligence constraint propagation techniques [4] can be employed in eliminating this enclosing quantifier.

First Section 2.1 reinterprets the enclosed conjunction as a *system of ordering constraints*. Then Section 2.2 explains how these constraints can be propagated, and how this propagation enables eliminating the enclosing existential quantifier. Finally Section 2.3 points out the cases where this propagation method is insufficient, and extends it to cover the missing cases as well.

2.1. Conjunctions as constraints

Let us extend the language L into L' by introducing the new relation symbols in Table 1. The first two were already in L, while the third reverses the direction of the ordering '$<$'.

The fourth, fifth and sixth new symbols are usually written as '$\leq$', '$\geq$' and '$\neq$', respectively. Our alternative notation shows that we treat these *compound symbols as sets of the first three basic symbols* '$<$', '$=$' and '$>$', as show in the final column. Accordingly the seventh compound symbol indicates that the variables x and y can relate to each other in any of these three basic ways. Finally, the last symbol indicates that they cannot be ordered with respect to each other in any consistent way.

Our constraint propagation approach employs this set-of-symbols representation in its symbolic manipulations. For instance, converting compound symbols to sets and back reveals the following:

Table 1
The relation symbols in language L'

L'-formula	Corresponding L-formula	As a set
$x < y$	$x < y$	$\{<\}$
$x = y$	$x = y$	$\{=\}$
$x > y$	$y < x$	$\{>\}$
$x \leqq y$	$x < y \vee x = y$	$\{<, =\}$
$x \geqq y$	$x = y \vee y < x$	$\{=, >\}$
$x \lessgtr y$	$x < y \vee y < x$	$\{<, >\}$
$x \leqq \geqq y$	$x < y \vee x = y \vee y < x$	$\{<, =, >\}$
$x \perp y$	$x < x \wedge y < y$	$\emptyset$

$$(x\ R\ y) \vee (x\ S\ y) \equiv_{\mathrm{DLO}} x\ (R \cup S)\ y \tag{8}$$

$$(x\ R\ y) \wedge (x\ S\ y) \equiv_{\mathrm{DLO}} x\ (R \cap S)\ y \tag{9}$$

$$\neg(x\ R\ y) \equiv_{\mathrm{DLO}} x\ \left(\begin{matrix}\leqslant \\ \geqslant\end{matrix} \setminus R\right)\ y. \tag{10}$$

Moreover,

$$y\ R\ x \equiv_{\mathrm{DLO}} x\ R'\ y \tag{11}$$

where R' is obtained from R by replacing each symbol '$<$' with '$>$' and vice versa.

Another reason to enrich language L into L' is to avoid considering every possible conjunction of atomic L-formulae separately. The number of such conjunctions over n variables is namely given by the nth ordered Bell number $\tilde{b}(n)$, or the number of ways to first divide n distinct items into groups (of equal variables) and then to order those groups. These numbers grow namely asymptotically as [12, Example 1 of Chapter 5.2]

$$\tilde{b}(n) \approx \frac{n!}{2(\log 2)^{n+1}} + \mathrm{O}(0.16^n n!). \tag{12}$$

2.2. Quantifier elimination via constraint propagation

We seek here the strongest possible conclusion we can draw about the relation between variables x_i and x_k, given what we already know about the relations between x_i and x_j on the one hand, and between x_j and x_k on the other hand.

Inference (in general, not just in DLO) is correct if and only if every model of its premises is also a model of its conclusion. That is, we must require that

$$\models_{\mathrm{DLO}} (x_i\ R\ x_j) \wedge (x_j\ R'\ x_k) \rightarrow (x_i\ S\ x_k) \tag{13}$$

in order for $(x_i\ S\ x_k)$ to be an acceptable conclusion from the premises $(x_i\ R\ x_j)$ and $(x_j\ R'\ x_k)$. If there is another acceptable conclusion $(x_i\ S'\ x_k)$ then also $\models_{\mathrm{DLO}} (x_i\ R\ x_j) \wedge (x_j\ R'\ x_k) \rightarrow (x_i\ S\ x_k) \wedge (x_i\ S'\ x_k)$, so $\models_{\mathrm{DLO}} (x_i\ R\ x_j) \wedge (x_j\ R'\ x_k) \rightarrow (x_i\ (S \cap S')\ x_k)$ by Eq. (9). Hence for every pair R and R' of relations there is a unique $R \otimes R'$ such that Eq. (13) holds exactly when $R \otimes R' \subseteq S$. Table 2 presents this operator in tabular form. This operator is what we sought: Anything more specific than $(x_i\ (R \otimes R')\ x_k)$ would no longer be warranted by $(x_i\ R\ x_j)$ and $(x_j\ R\ x_k)$, while anything less specific would lead to weaker conclusions with extra disjuncts when translated back to L.

Operator '$\otimes$' hints at our quantifier elimination strategy: $(x_i\ (R \otimes R')\ x_k)$ represents the information provided by $(x_i\ R\ x_j)$ and $(x_j\ R'\ x_k)$ directly, so the intermediate variable x_j is no longer needed and can be eliminated. However, incorporating this new information into what we already know about variables x_i and x_k might warrant further inference steps, and eventually elaborate what we know about variable x_j as well. Hence we must infer as much as we possibly can before attempting variable elimination.

Definition 2. Let ϕ be a conjunction of atomic L'-formulae and let F be the set of variables occurring in ϕ. Its *path consistent form* path(ϕ) is the closure of $\phi \wedge \bigwedge_{x_i \in F} (x_i = x_i)$ under the following two rules:

Table 2
The table for the operation $R \otimes R'$, with R as the rows and R' as the columns

$\otimes$	$\bot$	$<$	$=$	$>$	$\leqq$	$\geqq$	$\leqslant$	$\gtrless$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$<$	$\bot$	$<$	$<$	$\gtrless$	$<$	$\gtrless$	$\gtrless$	$\gtrless$
$=$	$\bot$	$<$	$=$	$>$	$\leqq$	$\geqq$	$\leqslant$	$\gtrless$
$>$	$\bot$	$\gtrless$	$>$	$>$	$\gtrless$	$>$	$\gtrless$	$\gtrless$
$\leqq$	$\bot$	$<$	$\leqq$	$\gtrless$	$\leqq$	$\gtrless$	$\gtrless$	$\gtrless$
$\geqq$	$\bot$	$\gtrless$	$\geqq$	$>$	$\gtrless$	$\geqq$	$\gtrless$	$\gtrless$
$\leqslant$	$\bot$	$\gtrless$	$\leqslant$	$\gtrless$	$\gtrless$	$\gtrless$	$\gtrless$	$\gtrless$
$\gtrless$	$\bot$	$\gtrless$	$\gtrless$	$\gtrless$	$\gtrless$	$\gtrless$	$\gtrless$	$\gtrless$

1. If ϕ contains both $(x_i\ R\ x_j)$ and $(x_i\ R'\ x_j)$ as conjuncts, then replace both of them with the single conjunct $(x_i(R \cap R')x_j)$ by Eq. (9).
2. If ϕ contains both $(x_i\ R\ x_j)$ and $(x_j\ R'\ x_k)$ as conjuncts, then add $(x_i\ (R \otimes R')\ x_k)$ as a new conjunct into ϕ by Table 2.

In both rules, if some conjunct $(x_p\ R\ x_q)$ is sought but $(x_q\ R'\ x_p)$ is available, we apply Eq. (11) first.

The function path(ϕ) in Definition 2 retains the logical meaning of its input formula ϕ, since in every application of rule 2 the added conjunct is an acceptable conclusion from its premises, by the construction leading to Table 2. We omit the details of such standard logical arguments in the interest of brevity, and focus only on those proofs that provide us with the building blocks required by our algorithm QF.

Logically rule 2 in Definition 2 updates what is currently known about the relation of variables x_i and x_k with additional information, namely the most specific conclusion we can draw from what we know about relations between x_i and x_j on the one hand and x_j and x_k on the other. This concept of *path* (*or* 3-) *consistency* with respect to binary constraints is of course well known in Artificial Intelligence [4, Chapter 3.3]: If we take any three variables x_i, x_j and x_k together with any three basic relation symbols $s_{pq} \in R_{pq}$ between them, then we can also find a DLO model $\mathfrak{M}$ and an assignment θ for which $\mathfrak{M} \models_{\mathrm{DLO}} (x_i\ s_{ij}\ s_j) \wedge (x_i\ s_{ik}\ s_k) \wedge (x_j\ s_{jk}\ s_k)\theta$.

The closure computation in Definition 2 is given in Fig. 1 as an incremental algorithm for adding a new conjunct into an existing conjunction which has already undergone the same computation. Technically, this algorithm deals only with those conjunctions ϕ that have at most one conjunct $(x_i\ R_{ij}\ x_j)$ for any index pair $i < j$, but extra conjuncts can be dealt with beforehand by applying Eqs. (9) and (11). Closure computation of a given conjunction ϕ then proceeds by incrementally adding each of its conjuncts into the identically true conjunction $\mathbf{T}$ without any real conjuncts. This overall computation takes $O(n^3)$ recursion calls: Each compound symbol R_{ij} can only shrink during the computation, so it

procedure PROPAGATE($x_i\ R'_{ij}\ x_j$) **is**
  if $R_{ij} \subsetneq R'_{ij}$ where $(x_i\ R_{ij}\ x_j)$ is the conjunct in ϕ **then**
    Replace R_{ij} with $R_{ij} \cap R'_{ij}$ in ϕ**;**
    for all conjuncts $(x_j\ R_{jk}\ x_k)$ in ϕ where k is neither i nor j **do**
      PROPAGATE$(x_i(R_{ij} \otimes R_{jk})x_k)$ recursively
    end for;
    for all conjuncts $(x_k\ R_{ki}\ x_i)$ in ϕ where k is neither i nor j **do**
      PROPAGATE$(x_k(R_{ki} \otimes R_{ij})x_j)$ recursively
    end for
  end if.

Whenever this algorithm requires a conjunct $(x_p\ R_{pq}\ x_q)$ but path(ϕ) provides $(x_q\ R_{qp}\ x_p)$, we assume a tacit application of Eq. (11) first. Similarly we assume that whenever this algorithm requires a conjunct $(x_p\ R_{pq}\ x_q)$ not in path(ϕ), the conjunct $(x_p \overset{\leq}{\underset{\geq}{}} x_q)$ is provided tacitly instead.

Fig. 1. Adding a new conjunct $(x_i\ R'_{ij}\ x_j)$ into path(ϕ) where $i \neq j$.

can change at most four times. Whenever a symbol R_{ij} changes into $R_{ij} \cap R'_{ij}$, this change propagates via its neighbours R_{ik} and R_{kj}, causing no more than $2(n-2)$ looping steps. Finally, there are $n(n+1)/2$ different shrinking symbols R_{ij}.

The closure computation in Definition 2 provides a check for the *satisfiability* of ϕ; that is, for the existence of a DLO model $\mathfrak{M}$ and an assignment θ such that $\mathfrak{M} \models \phi\theta$. As we shall need such a check later, let us next show that this is indeed so by considering the graph which encodes the ordering information in ϕ.

Definition 3. Let ϕ be a conjunction of atomic L'-formulae. Its *graph*, denoted graph(ϕ), is the following directed labelled multigraph.

Its nodes are equivalence classes of the variables in ϕ, where x_i and x_j are in the same node E if and only if the ϕ contains the conjunct $(x_i = x_j)$. Denote as rep(E) the unique variable $x_i \in E$ chosen arbitrarily as the representative for the entire class E.

Each edge is labelled with either '$<$' or '$\leqq$'. There is an edge $E \overset{R}{\rightarrow} E'$ if there are some $x_i \in E$ and $x_j \in E'$ for which ϕ contains the conjunct $(x_i\ R\ x_j)$.

Lemma 1. *Assume that* ϕ *is* path consistent; *that is,* ϕ *is the same formula as* path(ϕ) *in Definition* 2 (*modulo permutation of conjuncts*). *Then* graph(ϕ) *is a* transitively closed Directed Acyclic Graph (DAG); *that is, there is at most one edge from one node to another, and there is also a direct edge from node* E *to node* E' *whenever there is an indirect path from* E *to* E'*, and finally* E *is not* E' *in these paths.*

Proof. Path consistency implies that the choice of representatives is immaterial. More precisely, we have the following (modulo Eq. (11)) by the contents of the row and column labelled with '$=$' in Table 2:

$$(x_i\ R\ x_j) \text{ is in } \phi \quad \text{if and only if} \quad (\text{rep}(E)\ R\ \text{rep}(E')) \text{ is, where } x_i \in E \text{ and } x_j \in E'. \tag{14}$$

Let then $E \stackrel{R}{\to} E'$ and $E \stackrel{R'}{\to} E'$ be two edges in graph(ϕ). Then (rep(E) R rep(E')) and (rep(E) R' rep(E')) are two conjuncts between the same variables rep(E) and rep(E') in ϕ by Eq. (14). By rule 1 in Definition 2 we have $R = R'$. Hence there is at most one edge from one node E to another node E'.

Next assume that there is an indirect path $E_0 \stackrel{R_1}{\to} E_1 \stackrel{R_2}{\to} E_2 \stackrel{R_3}{\to} \cdots \stackrel{R_p}{\to} E_p$. Then ϕ contains the conjunct rep(E_0)($R_1 \cap R_2 \cap R_3 \cap \cdots \cap R_p$) rep($E_p$) by Eq. (14) and the contents of the rows and columns labelled with '$<$' and '$\leqq$' in Table 2. Hence graph(ϕ) contains also the corresponding direct edge

$$E_0 \xrightarrow{R_1 \cap R_2 \cap R_3 \cap \cdots \cap R_p} E_p \tag{15}$$

and is therefore transitively closed.

If finally $E_0 = E_p$ in Eq. (15), then the corresponding edge label $R_1 \cap R_2 \cap R_3 \cap \cdots \cap R_p$ is either '$\perp$' or '$=$' (depending on whether some R_q is '$<$' or not), and this contradicts the construction of graph(ϕ). Hence graph(ϕ) must be acyclic. □

Theorem 1. *Let ϕ be a path consistent conjunction of atomic L'-formulae. Then ϕ is unsatisfiable if and only if it consists of conjuncts of the form $x_i \perp x_j$.*

Proof. If ϕ contains a conjunct of the form $x_i \perp x_j$, then it is clearly unsatisfiable, since this conjunct is. Moreover, the closure computation in Definition 2 ensures that such a conjunct spreads '$\perp$' into all other conjuncts as well. Hence it suffices to assume that ϕ contains no such conjunct, and construct an assignment θ satisfying every conjunct in ϕ in the canonical DLO model, the real line $\mathbb{R}$, from Section 1.

Construct graph(ϕ) as in Definition 3. It is a DAG by Lemma 1. Hence its nodes can be topologically sorted into $E_1, E_2, E_3, \ldots, E_m$. We claim that setting $\theta(x_i) = p$ if $x_i \in E_p$ suffices. Consider any conjunct $(x_i\ R_{ij}\ x_j)$ in ϕ. If R_{ij} is '$=$', then x_i and x_j belong to the same equivalence class E_p, and so $\theta(x_i) = p = \theta(x_j)$ as required. If R_{ij} is '$<$' or '$\leqq$', then $x_i \in E_p$ and $x_j \in E_q$ where $p < q$, as permitted. The case where R_{ij} is '$>$' or '$\geqq$' is symmetric. If finally R_{ij} is '$\lesseqgtr$' or '$\stackrel{\leqq}{\geqq}$', then $x_i \in E_p$ and $x_j \in E_q$ where $p \neq q$, as permitted. □

Theorem 1 and the polynomiality of the algorithm in Fig. 1 noted above show that in this case satisfiability remains easy, whereas many extensions become NP-complete [5, Problem LO16].

However, quantifier elimination is not quite the same thing as satisfiability: In the latter we get to choose suitable values for *all* the variables at the same time. In the former suitable values are first given for *all except* the quantified variable, and only then do we get to choose a suitable value for the remaining quantified variable. Hence our main quantifier elimination theorem states only that a quantified variable can be eliminated after the constraint propagation in Definition 2 has been carried through, provided that the result of this propagation does not contain a specific forbidden subformula.

Theorem 2. *Let $\exists x_n.\phi$ be an L'-formula of the kind in Eq.* (7) *which satisfies the following three further assumptions*:

1. *The formula has free variables*; *that is*, $n > 1$.
2. *The conjunction* ϕ *is path consistent.*
3. *At least one of the following two conditions holds*:
   (a) *The conjunction* ϕ *has a variable* x_j *for which* ϕ *contains the conjunct* $(x_j = x_n)$.
   (b) *The conjunction* ϕ *has no subformula* $(x_i \leqq x_n) \wedge (x_n \leqq x_j) \wedge (x_i \leqq x_j) \wedge (x_k \lesseqgtr x_n)$ *even after reordering conjuncts and applications of Eq.* (11).

Then $\mathrm{QF}(\exists x_n.\phi) = \mathrm{drop}(x_n, \phi)$ *where* $\mathrm{drop}(x_n, \phi)$ *is obtained from* ϕ *by erasing each conjunct that contains variable* x_n.

Proof. Formula $\exists x_n.\phi$ can be reordered (by applications of familiar FOPC equivalences) into the form $\mathrm{drop}(x_n, \phi) \wedge \exists x_n. \bigwedge_{1 \leqslant i \leqslant n} (x_i \ R_{in} \ x_n)$. Hence it suffices (by FOPC truth definition) to assume that $\mathfrak{M} \models \mathrm{drop}(x_n, \phi)\theta$, and then find some $a \in \mathcal{U}_{\mathfrak{M}}$ such that $\mathfrak{M} \models (x_i \ R_{in} \ x_n)\theta[x_n/a]$ for each i; that is, to find a value a for the variable x_n which satisfies every atomic formula involving x_n in ϕ. The first half of this assumption implies that no R_{in} is '$\perp$' by Theorem 1.

If assumption 3(a) holds, then it suffices to choose $a = \theta(x_j)$: As above, assumption 2 ensures that each $R_{in} = R_{ij}$. Suppose therefore that assumption 3(a) fails. Then each R_{in} must contain at least one of '$<$' or '$>$', and assumption 3(b) must hold.

Let $\mathcal{B} = \{\theta(x_i)\colon\ 1 \leqslant i < n\}$; this set is nonempty by assumption 1. If each R_{in} contains '$>$', then it suffices to choose any $a <_{\mathfrak{M}} \min \mathcal{B}$ and we are done: Such arbitrarily small values exist by DLO axiom (5). Similarly, if each R_{in} contains '$<$', then any $\max \mathcal{B} <_{\mathfrak{M}} a$ suffices, and these exist by DLO axiom (6). Assume therefore that exist indices i for which '$>$' is not in R_{in}, and indices i' for which '$<$' is not in $R_{i'n}$.

Let us verify the following fact: If $\theta(x_j) <_{\mathfrak{M}} \theta(x_i)$ but '$>$' is not in R_{in}, then '$<$' is in R_{jn}. Otherwise we would have the following situation: First, R_{ni} is either '$>$' or '$\geqq$', since we saw above that it must contain at least one of '$>$' and '$<$', and the latter option is now ruled out. Second, R_{jn} is either '$>$' or '$\geqq$', by similar reasoning. And third, R_{ji} must contain '$<$', since $\theta(x_j) <_{\mathfrak{M}} \theta(x_i)$. But this violates path consistency, since the '$<$' can be removed from R_{ji}. Hence the fact is verified.

Let $l = \max\{\theta(x_i)\colon$ '$>$' is not in $R_{in}\}$. The case where this set would be empty has already been treated above. Then the fact shown above implies the following: First, we cannot choose any $a <_{\mathfrak{M}} l$, since it would cause conjunct $(x_i \ R_{in} \ x_n)$ to fail. Second, we have '$<$' in each R_{jn} where $\theta(x_j) <_{\mathfrak{M}} l$, as this is the conclusion of the fact. And third, we have '$>$' in each R_{kn} where $l <_{\mathfrak{M}} \theta(x_k)$, by the maximality of l.

Symmetrically to this construction, we can also find $r = \min\{\theta(x_j)\colon$ '$<$' is not in $R_{jn}\}$. We see again that we cannot choose $r <_{\mathfrak{M}} a$, that '$<$' is in each $R_{j'n}$ where $\theta(x_{j'}) <_{\mathfrak{M}} r$, and that '$>$' is in each $R_{k'n}$ where $r <_{\mathfrak{M}} \theta(x_{k'})$.

In combination, we see that we must choose our a from the closed interval $\mathcal{C} = [l, r]$. Moreover, all points other than those in $\mathcal{B}$ are surely sufficient.

This interval $\mathcal{C}$ is nonempty, since $r <_{\mathfrak{M}} l$ is impossible by the construction. Fig. 2 illustrates the current state of the construction.

If $\mathcal{C}$ is not a singleton, then sufficient points exist by DLO axiom (4).

It remains to check the case where $\mathcal{C}$ is a singleton; that is, $a = l = r$ is our only choice. This choice is also possible unless there exists some index $1 \leqslant h < n$ such that $\theta(x_h) = a$

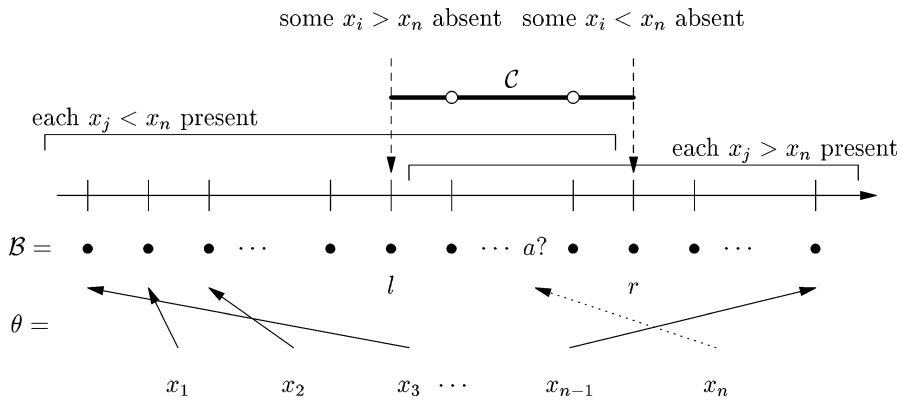

Fig. 2. The punctured interval $\mathcal{C}$ into which $a = \theta(x_n)$ should belong.

but R_{hn} does not contain '='. Choose some x_i such that $\theta(x_i) = a$ but '>' is not in R_{in}; the construction of l ensures they exist. Choose similarly some x_j such that $\theta(x_j) = a$ but '<' is not in R_{jn} from the construction of r. Then '=' must be in R_{ij} since $\theta(x_i) = a = \theta(x_j)$. Neither R_{in} nor R_{jn} can contain just '=', since assumption 3(a) has already been considered separately above, so R_{in}, R_{nj} and R_{ij} must all be '$\leqq$' by path consistency. Then assumption 3(b) implies that there is no index $1 \leqslant k < n$ for which R_{kn} would be '$\leqslant$'. On the other hand, if R_{hn} is just '<', then R_{hj} is also just '<' by path consistency and the value inferred above for R_{nj}, but this contradicts $\theta(x_h) = a = \theta(x_j)$. Similarly, value '>' for R_{hn} contradicts R_{in}. Hence such an index h cannot exist. □

Consider then the assumptions of Theorem 2. If assumption (1) fails, then the formula is of the form $\exists x_1.\phi'$ where ϕ' is a conjunction of atomic L'-formulae over just the single variable x_1. Then ϕ' can be simplified into just one conjunct $(x_1\ R\ x_1)$ by repeated applications of Eq. (9). Then Theorem 2 extends to this case if we define

$$\mathrm{drop}(x_1, x_1\ R\ x_1) = \begin{cases} \mathbf{T} & \text{if '=' is in } R \\ \mathbf{F} & \text{otherwise} \end{cases} \tag{16}$$

where '**F**' denotes the identically false variable-free formula. In the following quantifier elimination computations **F** behaves like a disjunction without any disjuncts, complementary to the identically true quantifier-free formula **T** which was defined above to behave as a conjunction without conjuncts. At the end of the day **T** and **F** can be written out as $(x_0 = x_0)$ and $(x_0 \lesseqgtr x_0)$, respectively, where we reserve the variable symbol x_0 for this purpose.

Assumption 2 can be enforced with the closure computation of Definition 2. Finally, assumption 3 is discussed in Section 2.3 below.

2.3. Shortcomings of constraint propagation

Here we consider the only situation where Theorem 2 cannot be applied: that is, there is no conjunct required by condition 3(a), but there is a subformula forbidden by condition 3(b). In such a situation, we can proceed by considering what else is known about

the relations between the variables involved in the forbidden subformula. This leads to the following case-by-case analysis.

Consider first the case where in addition to the forbidden subformula of condition 3(b) the conjunction ϕ also has $(x_i \leqq x_k) \wedge (x_k \leqq x_j)$. Intuitively, both x_n and x_k are now constrained to lie in the interval $\mathcal{C}$ shown in Fig. 2. On the other hand, $(x_n \lesseqgtr x_k)$, so this $\mathcal{C}$ cannot be a singleton interval. And indeed,

$$\begin{aligned} \models_{\mathrm{DLO}} (x_i \leqq x_n) \wedge (x_n \leqq x_j) \wedge (x_i \leqq x_j) \wedge (x_k \lesseqgtr x_n) \wedge (x_i \leqq x_k) \\ \wedge (x_k \leqq x_j) \rightarrow (x_i < x_j), \end{aligned} \tag{17}$$

so by the discussion in the beginning of Section 2.2 we can add the rule corresponding to Eq. (17) to the path consistency computation of Definition 2. Let us call the result of this enriched closure computation the *apex* consistent form $\mathrm{apex}(\phi)$ of the conjunction ϕ.

As in Definition 2 we see that the function apex retains the logical meaning of its input formula ϕ. Assumption 2 of Theorem 2 can also be strengthened from path to apex consistency.

The reason for moving from path to apex consistency is that Eq. (17) involves *four* variables, whereas Definition 2 involved only three. However, this fourth variable is always fixed to be the variable x_n being eliminated. Hence apex consistency is a weaker property than full 4-consistency [4, Chapter 3.4]. Apex consistency can be computed within the same $\mathrm{O}(n^3)$ steps as path consistency via a straightforward extension to the algorithm in Fig. 1, since the fourth variable x_n to consider is fixed.

Consider on the other hand the following calculation:

$$\exists x_4.(x_1 \leqq x_4) \wedge (x_4 \leqq x_2) \wedge (x_1 \leqq x_2) \wedge (x_3 \lesseqgtr x_4) \tag{18}$$

$$\begin{aligned} \equiv_{\mathrm{DLO}} &\big(\exists x_4.(x_1 = x_4) \wedge (x_4 \leqq x_2) \wedge (x_1 \leqq x_2) \wedge (x_3 \lesseqgtr x_4)\big) \vee \\ &\big(\exists x_4.(x_1 < x_4) \wedge (x_4 \leqq x_2) \wedge (x_1 \leqq x_2) \wedge (x_3 \lesseqgtr x_4)\big) \end{aligned} \tag{19}$$

$$\begin{aligned} \equiv_{\mathrm{DLO}} &(x_1 \leqq x_2) \wedge (x_3 \lesseqgtr x_1) \vee \\ &(x_1 < x_2). \end{aligned} \tag{20}$$

Here the initial formula (18) is the simplest example of the case where ϕ has

$$\big(x_i \overset{\leqq}{\underset{>}{\leqq}} x_k\big) \wedge \big(x_k \overset{\leqq}{\underset{>}{\leqq}} x_j\big)$$

instead of the previous case. Formula (19) is then obtained by first breaking $x_1 \leqq x_4$ into disjuncts as permitted by Table 1, and then applying familiar FOPC equivalences. Finally formula (20) is obtained by applying Theorem 2 to each disjunct separately now that its assumption 3 has taken hold in each of them.

However, the resulting quantifier-free formula (20) is no longer expressible as a conjunction of atomic L'-formulae: In any such conjunction, the relation R_{12} between x_1 and x_2 must be '$\leqq$', but what would then be the relation R_{13} between x_1 and x_3? Moreover, the remaining cases where ϕ has either

$$(x_i \leqq x_k) \wedge \big(x_k \overset{\leqq}{\underset{>}{\leqq}} x_j\big) \quad \text{or} \quad \big(x_i \overset{\leqq}{\underset{>}{\leqq}} x_k\big) \wedge (x_k \leqq x_j)$$

lead to similar calculations. Hence we see that *it would be useless to go beyond apex consistency* of two-variable constraints, since in all these other cases we obtain disjunctions of conjunctions.

On the other hand, formulae (18)–(20) do point out a strategy to reduce these problematic cases into manageable ones: Select some conjunct $(x_i\ R_{ij}\ x_j)$ where the set R_{ij} is not a singleton, break R_{ij} into the corresponding disjunction involving two nonempty sets R and S by Eq. (8), and perform further quantifier elimination on each of the resulting disjuncts separately. This strategy terminates eventually, because both R and S are proper subsets of R_{ij}.

This strategy must somehow select this conjunct to break. We suggest the following straightforward *greedy heuristic:* Pick some x_i such that

1. x_i belongs to some violation $(x_i \leqq x_n) \wedge (x_n \leqq x_j) \wedge (x_i \leqq x_j) \wedge (x_k \lesseqgtr x_n)$ of assumption 3(b) of Theorem 2, and
2. x_i has a maximal number of such $x_{i'}$ that $x_{i'} \leqq x_i$ and $(x_{i'} \leqq x_n) \wedge (x_n \leqq x_{j'}) \wedge (x_{i'} \leqq x_{j'}) \wedge (x_{k'} \lesseqgtr x_n)$ is some other such violation.

Then select $(x_i \leqq x_n)$ as the conjunct to break.

This selection breaks into $(x_i = x_n) \vee (x_i < x_n)$, as in formula (19). We argue informally below that both of these two disjuncts are effective in enabling subsequent use of Theorem 2.

Consider first the disjunct $(x_i = x_n)$. It satisfies assumption 3(a) directly. No further conjuncts need to be broken.

Consider then the other disjunct $(x_i < x_n)$. It removes not only the violation of assumption 3(b) in heuristic condition 1 but also each violation in heuristic condition 2 as well: From it and $(x_{i'} \leqq x_i)$ the path consistency computation in Definition 2 namely concludes $(x_{i'} < x_n)$. Hence we greedily maximize the number of such removals, hoping that this will reduce the need to break more conjunctions.

By symmetry, we could also have selected the opposite conjunct $(x_n \leqq x_j)$ in condition 1. Even better, our heuristic could choose dynamically between these x_i and x_j based on which yields now a higher number of $x_{i'}$ and $x_{j'}$ in condition 2.

What about the other choices available for our heuristic? We could have chosen $(x_i \leqq x_j)$ instead, but changes to $(x_i \leqq x_n) \wedge (x_n \leqq x_j)$ imply changes to this choice as well. Hence the proposed heuristic attains the same benefits already. Or we could have chosen $(x_k \lesseqgtr x_n)$ instead and break it into $(x_k < x_n) \vee (x_k > x_n)$, but in at least one of these two disjuncts only this particular violation involving x_k disappears, since by apex consistency at least one of $\left(x_i \lesseqgtr x_k\right)$ and $\left(x_k \lesseqgtr x_j\right)$ is also present. Hence the proposed heuristic is never worse, and might be better.

An entirely different approach would be to somehow select some conjunct $(x_p\ R_{pq}\ x_q)$ which is not directly involved in any violation but which nevertheless causes the disappearance of many violations when broken. However, this would require a selection rule which can somehow take into account these indirect dependencies caused by the path and apex consistency computations without being overly complicated. Theorem 3 below shows one possible starting point, but we do not pursue this approach here.

Another approach not pursued here would be to rearrange the order in which variables are to be eliminated: If the conjunction ϕ is prefixed by several quantifiers, as in $\exists x_{n-1}.\exists x_n.\phi$, then a better result might be obtainable by starting with the outer variable x_{n-1} instead of the inner variable x_n.

In summary, a given conjunction ϕ of atomic L'-formulae can be converted into a disjunction of such conjunctions that meet all the assumptions of Theorem 2, and are therefore amenable to quantifier elimination. Thus we have

$$\mathrm{QF}(\exists x_n.\phi) = \mathrm{break}\big(\mathrm{apex}(\phi)\big) \quad \text{where} \tag{21}$$

$$\mathrm{break}(\psi) = \begin{cases} \mathrm{drop}(x_n, \psi) & \text{if } \psi \text{ meets the assumptions} \\ \mathrm{path}(\psi \wedge (x_i = x_n)) \vee & \\ \mathrm{break}(\mathrm{apex}(\psi \wedge (x_i < x_n))) & \text{otherwise, with } x_i \text{ from heuristic.} \end{cases} \tag{22}$$

(We also assume that Eq. (9) is applied before the path or apex consistency computation in Eq. (22).) We moreover argued that Eq. (22) should generate few disjuncts.

3. From conjunctions to all formulae

Eq. (21) solved the quantifier elimination problem for all formulae of the form (7), where an existential quantifier presided over a conjunction on atomic L'-formulae. Here we extend this solution to all L'-formulae.

It is enough to concentrate on such L'-formulae that are built from the atomic formulae with conjunction, disjunction, and possibly negated existential quantification: Negations can be eliminated from atomic formulae with Eq. (10), while standard FOPC manipulations suffice for the rest.

Our first quantification elimination algorithm simply generates an equivalent disjunction of conjunctions of atoms with familiar equivalences of propositional logic, and Eqs. (10) and (21).

$$\mathrm{QF}(\phi) = \phi \quad \text{if } \phi \text{ is an atomic } L'\text{-formula} \tag{23}$$

$$\mathrm{QF}(\phi \vee \psi) = \mathrm{QF}(\phi) \vee \mathrm{QF}(\psi) \tag{24}$$

$$\mathrm{QF}(\phi \wedge \psi) = \mathrm{conj}\big(\mathrm{QF}(\phi), \mathrm{QF}(\psi)\big) \tag{25}$$

$$\mathrm{conj}(\phi, \psi) = \bigvee_{\substack{\phi' \text{ is a conjunction in } \phi \\ \psi' \text{ is a conjunction in } \psi}} \phi' \wedge \psi' \tag{26}$$

$$\mathrm{QF}(\exists x_n.\phi) = \bigvee_{\phi' \text{ is a conjunction in } \mathrm{QF}(\phi)} \mathrm{break}\big(\mathrm{apex}(\phi')\big) \tag{27}$$

$$\mathrm{QF}(\neg\phi) = \mathrm{neg}(\mathrm{QF}(\phi)) \tag{28}$$

$$\mathrm{neg}(\phi) = \begin{cases} \bigvee_{(x_i\ R_{ij}\ x_j) \text{ is in } \phi} \big(x_i \big(\lesseqgtr \backslash R_{ij}\big) x_j\big) & \text{if } \phi \text{ is a conjunction} \\ \mathrm{conj}(\mathrm{neg}(\phi'), \mathrm{neg}(\phi'')) & \text{if } \phi \text{ is } \phi' \vee \phi'' \\ \mathbf{T} & \text{if } \phi \text{ is } \mathbf{F}. \end{cases} \tag{29}$$

However, this algorithm can be improved in several ways. Section 3.1 improves the way it handles negation, while Section 3.2 improves its memory requirements.

3.1. Negation for less

Eq. (26) is computationally expensive, since it builds a conjunction of two formulae which are given in *Disjunctive Normal Form* (*DNF*), and this involves a quadratic blow-up in formula length. It would therefore be worthwhile to minimize these two formulae first. Unfortunately, minimizing propositional formulae in DNF is Σ_2^{P}-complete [11, Theorem 4], and remains so even if only deleting literals (propositional symbols and their negations) is allowed [11, Problem TERM-WISE MIN DNF].

What can be done is to minimize each conjunction ψ in $\mathrm{QF}(\phi)$ separately by omitting from ψ each conjunct which can be inferred from the other conjuncts in ψ with the path consistency computation of Definition 2. It would be pointless to include apex consistency as well, since this negation being computed is in front of an existential quantifier, and therefore the fourth variable of apex consistency has just been eliminated. For the same reason, we can assume ψ to be path consistent to begin with. (It is easy to see that the operation 'drop' preserves path consistency.)

This minimization does not really improve the use of Eq. (26) in Eq. (25). However, it does improve its use in recurrence (29) by minimizing the disjunction of atomic L'-formulae in the conjunction case of the recurrence, and is therefore worthwhile.

Theorem 3. *Let ψ be a path consistent conjunction of atomic L'-formulae. Then $\psi \equiv_{\mathrm{DLO}} \mathrm{red}(\psi)$ where the* reduction $\mathrm{red}(\psi)$ *of ψ is computed as follows*:

1. *If ψ is unsatisfiable as in Theorem* 1, *then let* $\mathrm{red}(\psi)$ *be* **F** *and stop.*
2. *Otherwise build* $\mathrm{graph}(\psi)$ *as in Definition* 3 *and continue as follows.*
3. *Mark an edge $E \overset{R}{\to} E''$ if and only if there exists an edge pair $E \overset{R'}{\to} E' \overset{R''}{\to} E''$ such that $R = R' \otimes R''$ in Table* 2.
4. *Let*

$$\mathrm{red}(\psi) \text{ be } \bigwedge_{\substack{E \text{ is a node in } \mathrm{graph}(\psi),\\ x_i \in E \setminus \{\mathrm{rep}(E)\}}} \big(x_i = \mathrm{rep}(E)\big) \tag{30}$$

$$\wedge \bigwedge_{E \overset{R}{\to} E' \text{ is an unmarked edge in } \mathrm{graph}(\psi)} \big(\mathrm{rep}(E)\ R\ \mathrm{rep}(E')\big) \tag{31}$$

$$\wedge \bigwedge_{\substack{E \text{ and } E' \text{ are nodes in } \mathrm{graph}(\psi),\\ (\mathrm{rep}(E) \leqslant \mathrm{rep}(E')) \text{ is a conjunct in } \psi}} \big(\mathrm{rep}(E) \leqslant \mathrm{rep}(E')\big) \tag{32}$$

and stop.

Proof. If the computation of $\mathrm{red}(\psi)$ stops at step 1, then the claim follows directly. Assume then that ψ is path consistent, and so $\mathrm{red}(\psi)$ is computed in step 4.

Then $\mathrm{graph}(\psi)$ constructed in step 2 is a transitively closed DAG by Lemma 1. Moreover, each conjunct in $\mathrm{red}(\psi)$ is also in ψ (modulo Eq. (11)) by Eq. (14). Hence it suffices to show that each conjunct $(x_i\ R_{ij}\ x_j)$ that is in ψ but is omitted from $\mathrm{red}(\psi)$ is implied by $\mathrm{red}(\psi)$ (again modulo Eq. (11)).

Let $x_i \in E_i$ and $x_j \in E_j$, and note that conjunction (30) implies $(x_k = \mathrm{rep}(E_k))$ for all $x_k \in E_k$.

If R_{ij} is '$\overset{\leqq}{\geqq}$', then the result holds trivially.

If R_{ij} is '$=$', then $E_i = E_j$ by the construction of graph(ψ), from which the result follows.

If R_{ij} is '$<$' or '$\leqq$', then we reason as follows (cases '$>$' and '$\geqq$' being symmetric modulo Eq. (11)). If we ignore the edge labels for a moment, we see that step 3 computes the *transitive reduction* of the transitively closed graph(ψ) [2, Section 2]. If we then consider the extra condition on the labels, we see that we mark an edge only when not only the edge but also its label can be generated from the labels of the corresponding edge pair during subsequent transitive closure computation. Hence conjunction (31) implies $(\mathrm{rep}(E_i)\ R_{ij}\ \mathrm{rep}(E_j))$ by the transitivity of relations '$<$' and '$\leqq$', and the result follows.

If finally R_{ij} is '$\lesseqgtr$', then conjunction (32) contains $(\mathrm{rep}(E_i) \lesseqgtr \mathrm{rep}(E_j))$, from which the result follows. □

Theorem 3 lets us replace Eq. (29) with

$$\mathrm{neg}(\phi) = \begin{cases} \mathrm{inv}(\mathrm{red}(\phi)) & \text{if } \phi \text{ is a conjunction} \\ \mathrm{conj}(\mathrm{neg}(\phi'), \mathrm{neg}(\phi'')) & \text{if } \phi \text{ is } \phi' \vee \phi'' \\ \mathbf{T} & \text{if } \phi \text{ is } \mathbf{F} \end{cases} \tag{33}$$

$$\mathrm{inv}(\psi) = \begin{cases} \mathbf{T} & \text{if } \psi \text{ is } \mathbf{F} \\ \bigvee_{(x_i\ R_{ij}\ x_j) \text{ is in } \psi} \left(x_i \left(\overset{\leqq}{\geqq} \setminus R_{ij}\right) x_j\right) & \text{otherwise.} \end{cases} \tag{34}$$

Note finally that the reduced form $\mathrm{red}(\psi)$ of a conjunction ψ is useful not only in Eq. (34) above but also in printing out the disjuncts of the final result of the entire quantifier elimination computation.

3.2. The incremental algorithm

Let us then seek an algorithm which computes its output disjunction *incrementally* one disjunct at a time. Our aim is to reduce the memory consumption of the algorithm by always generating the next disjunct β only when β is really needed for carrying out the next operation e: If β is needed only for e, then the memory occupied by β can be reclaimed for other use after e. To this end, we assume that whenever a disjunction of the form $\alpha \vee \beta$ appears on the right hand side of our algorithmic equations then its right operand β is evaluated *lazily* [10, Chapter 11.2.2]. In fact, our algorithm evaluates this β eventually (at most) once, so it suffices to use a thunk $\lambda\_.\beta$ to delay the evaluation of β; memoizing the value of β is not necessary. Moreover, we consider the lazily generated disjunction as a lazy right associative conjunction list of the form $\alpha \vee (\beta \vee (\gamma \vee (\ldots \vee \mathbf{F}))\ldots)$ but suppress the concomitant list operations in our equations below.

In this functional setting the algorithm in Fig. 1 must naturally use appropriate functional data structures instead of imperative replacement, such as persistent red-black trees [9, Chapter 3.3] with a concomitant slowdown logarithmic in the number n of variables.

However, Eq. (26) presents again a problem, just as it did in Section 3.1: The result consists of pairing together a disjunct ϕ' of ϕ and another disjunct ψ' of ψ in all possible

ways. Hence ϕ' and ψ' cannot be discarded straight after they have been used, as suggested above.

Therefore we develop instead a *two-argument* function $\mathrm{QF}'(\phi, \chi)$, where χ is a conjunction of atomic L'-formulae, which computes a quantifier-free formula DLO-equivalent to $\phi \wedge \chi$. This second argument χ permits us to replace Eqs. (25) and (26) with

$$\mathrm{QF}'(\phi \wedge \psi, \chi) = \bigvee_{\psi' \text{ is a conjunction in } \mathrm{QF}'(\psi,\chi)} \mathrm{QF}'(\phi, \psi'). \tag{35}$$

This form is in turn computable incrementally, since we can generate the first disjunct(s) of the result as soon as we have generated the first disjunct ψ'' of $\mathrm{QF}'(\psi, \chi)$ with $\mathrm{QF}'(\phi, \psi'')$, and so on. The disjunction in Eq. (35) should be read as a right associative sequence of appropriate lazy list append operations to retain the list form prescribed above. The other disjunctions below should also be read similarly.

In fact, using *higher-order functions* [10, Chapter 11.2.3] leads us to consider Eq. (35) as applying the function $f_\phi = \lambda v.\, \mathrm{QF}'(\phi, v)$ to each conjunction ψ' produced by $\mathrm{QF}'(\psi, \chi)$. This f_ϕ acts then as a *distiller* which takes in a conjunction v and produces those subformulae of v which are consistent with ϕ as well. This *staging* idea can be taken even further: the algorithm below can be considered a *compiler* $\lambda u.\, f_u$ which converts a given formula u into the corresponding distiller f_u. However, we shall refrain from doing so below in the interest of notational clarity.

Adding this second argument χ to Eqs. (23), (24) and (27) is a straightforward application of familiar propositional equivalences:

$$\mathrm{QF}'(\phi, \chi) = \phi \wedge \chi \quad \text{if } \phi \text{ is an atomic } L'\text{-formula} \tag{36}$$

$$\mathrm{QF}'(\phi \vee \psi, \chi) = \mathrm{QF}'(\phi, \chi) \vee \mathrm{QF}(\psi, \chi) \tag{37}$$

$$\mathrm{QF}'(\exists x_n.\phi, \chi) = \bigvee_{\phi' \text{ is a conjunction in } \mathrm{QF}'(\phi,\chi)} \mathrm{break}\big(\mathrm{apex}(\phi')\big). \tag{38}$$

In Eq. (38) we must (as usual) assume that the quantified variable x_n does not appear in χ to avoid inadvertent capture. This is easily accomplished as a preprocessing step which *renames* (by selecting a fresh index) all quantified variables apart from each other and the free variables in the whole formula. Hence our original function $\mathrm{QF}(\phi)$ becomes

$$\mathrm{QF}(\phi) = \mathrm{QF}'\big(\mathrm{rename}(\phi), \mathbf{T}\big). \tag{39}$$

Handling negation remains. If we apply the idea behind Eq. (35) to Eq. (33) then the result is as follows:

$$\mathrm{QF}'\big(\neg\phi, \chi\big) = \mathrm{neg}'\big(\mathrm{QF}'(\phi, \chi), \chi\big) \tag{40}$$

$$\mathrm{neg}'(\phi, \chi) = \begin{cases} \mathrm{inv}'(\mathrm{red}(\phi), \chi) & \text{if } \phi \text{ is a conjunction} \\ \displaystyle\bigvee_{\psi' \text{ is a conjunction in } \mathrm{neg}'(\phi'',\chi)} \mathrm{neg}'(\phi', \psi') & \text{if } \phi \text{ is } \phi' \vee \phi'' \\ \chi & \text{if } \phi \text{ is } \mathbf{F} \end{cases} \tag{41}$$

$$\mathrm{inv}'(\psi, \chi) = \begin{cases} \chi & \text{if } \psi \text{ is } \mathbf{F} \\ \displaystyle\bigvee_{(x_i\ R_{ij}\ x_j) \text{ is in } \psi} \big(\chi \wedge \big(x_i \big(\tfrac{\leqq}{\geqq} \setminus R_{ij}\big) x_j\big)\big) & \text{otherwise.} \end{cases} \tag{42}$$

However, incrementality is not achieved, since the post-processing of the recursively obtained seemingly lazy disjunction $\mathrm{QF}'(\phi, \chi)$ generates all its disjuncts before yielding even the first disjunct of the result of $\mathrm{QF}'(\neg\phi, \chi)$.

Fortunately Eq. (42) suggests a different strategy: Whereas the other branches of the algorithm have proceeded recursively with respect to the first formula, here it can proceed recursively with respect to the second formula χ. This is because $R_{ij} \subseteq R'_{ij}$ where $(x_i\ R'_{ij}\ x_j)$ is the corresponding atomic formula in χ, since ψ is a disjunct in $\mathrm{QF}'(\phi, \chi)$. Moreover, it suffices to construct the disjunction over just those R_{ij} for which $R_{ij} \subsetneq R'_{ij}$, since if $R_{ij} = R'_{ij}$ then the corresponding disjunct is $\mathbf{F}$. That is, the following replacement to Eqs. (40)–(42) is correct and terminating:

$$\mathrm{QF}\big(\neg\phi, \chi\big) = \mathrm{neg}'\big(\neg\phi, \mathrm{QF}'(\phi, \chi), \chi\big) \tag{43}$$

$$\mathrm{neg}'(\neg\phi, \phi', \chi) = \begin{cases} \mathrm{inv}'(\neg\phi, \mathrm{red}(\psi), \mathrm{path}(\chi)) & \text{if } \psi \text{ is any satisfiable disjunct in } \phi' \\ \chi & \text{if } \phi' \text{ has no other disjuncts than } \mathbf{F} \end{cases} \tag{44}$$

$$\mathrm{inv}'(\neg\phi, \psi, \chi) = \bigvee_{\substack{(x_i\ R_{ij}\ x_j) \text{ is in } \psi, \\ (x_i\ R'_{ij}\ x_j) \text{ is in } \chi, \\ R_{ij} \subsetneq R'_{ij}}} \mathrm{QF}'\big(\neg\phi, \chi \wedge \big(x_i (R'_{ij} \setminus R_{ij}) x_j\big)\big). \tag{45}$$

Eqs. (43)–(45) do in turn achieve incrementality as follows. Eq. (44) can scan the result list of $\mathrm{QF}'(\phi, \chi)$ incrementally until the first satisfiable conjunction ψ is found (if any). Then the remaining unscanned tail part of ϕ' is discarded. Lazy evaluation has spent no computational effort in constructing this part (save for the cost of generating the appropriate thunk). Then Eq. (45) can easily form the resulting disjunction of recursive call results incrementally. However, whereas iteration in Eqs. (35)–(42) involved merely list processing, Eq. (45) involves full fixpoint recursion. This sin will return to haunt us later in Section 4.

The total depth of the recursion caused by negations is at most quadratic in the number n of variables in the whole formula in Eqs. (43)–(45) by the analysis of the algorithm in Fig. 1, whereas the space consumption in Eqs. (41) and (42) was dependent on the total number of disjuncts in the fully constructed subresult of $\mathrm{QF}'(\phi, \chi)$, and this could have been much larger by Eq. (12). This ensures that our final algorithm works in polynomial space with respect to the length of the whole formula, provided that the overall result is extracted from Eq. (39) incrementally one disjunct at a time.

Our final algorithm works in singly exponential time with respect to the length of the whole formula: On the one hand, its polynomial space complexity derived above ensures that its time complexity cannot be higher than this. On the other hand, for example given the input formula $\exists x_{4n}. \bigwedge_{1 \leqslant i \leqslant n} (x_i < x_{i+n} \vee x_{i+2n} < x_{i+3n})$ our algorithm converts the subformula within the quantification from Conjunctive to Disjunctive Normal Form, and this latter form has 2^n different disjuncts.

This concludes our incremental algorithm given as Eqs. (22), (35)–(39), and (43)–(45). However, let us add two minor remarks. First, the incremental algorithm in Fig. 1 is eminently suitable for Eqs. (22), (36) and (45), the places where new information is actually added to the result. And second, by this first remark both the input χ and the resulting formula of QF' are always path consistent.

4. Conclusion

Now that we have developed our quantifier elimination algorithm for the theory of dense linear order, let us discuss the application area we have in mind for it, namely *constraint databases* [1, pp. 94–96]. The logical view to classical relational databases is that a query ϕ specifies a condition on its free variables $x_1, x_2, x_3, \ldots, x_n$ and the answer to ϕ is the set of all such tuples $\tau = \langle a_1, a_2, a_3, \ldots, a_n \rangle$ that assigning a_1 to x_1, a_2 to x_2, a_3 to x_3 and so on satisfies ϕ. Constraint databases extend this view by replacing the tuples τ with formulae ψ_τ which do not give explicit elements $a_1, a_2, a_3, \ldots, a_n$ but merely express the constraints these elements must fulfill. One classical answer tuple τ then corresponds to the formula $x_1 = a_1 \wedge x_2 = a_2 \wedge x_3 = a_3 \wedge \cdots \wedge x_n = a_n$ whereas in the general case the relations in ψ_τ could be more elaborate than mere equality. Similarly, the set of tuples ϕ is replaced with a disjunction of formulae ψ_τ. Here quantifier elimination is a key step in query evaluation, since the answer should clearly consist of much simpler formulae than the query itself. Indeed, constraint databases are an eminent application area for a quantifier elimination algorithm over real closed fields [3], since constraint databases require arithmetic on real numbers rather than merely ordering constraints.

This view fits our algorithm in Section 3.2 well: Its second argument χ represents the constraint database as a disjunction of conjunctions of individual constraints as above. Hence the staged form of our algorithm is in fact a query language compiler which translates a given query ϕ into a distiller f_ϕ which in turn refines any given constraint database χ into a corresponding answer $f_\phi(\chi)$ of the same form.

Our algorithm further suggests that such constraint database query languages could be naturally embedded into *functional database query languages* [6]. In this setting, incremental computation ensures that the answer to a query can be computed without storing large intermediate results into the database. And indeed, our compiler does translate the given query ϕ into an incremental and purely functional program f_ϕ. However, while list processing is a basic programming technique in these query languages, the full fixpoint recursion in Eq. (45) is undesirable, so our construction does not quite achieve such an embedding.

This leads to our two open questions: First, can negation (or equivalently universal quantification) be handled efficiently with just list processing without full recursion? And second, more importantly, how incrementally can quantifier elimination be performed in the general setting of real closed fields? For instance, our construction might be of use, if a more elaborate constraint concept than in Section 2 is employed and the requisite results, especially an analogue to Theorem 2, are derived for it.

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, Reading, MA, 1995.
[2] A. Aho, M. Garey, J. Ullman, The transitive reduction of a directed graph, SIAM J. Comput. 1 (2) (1972) 131–137.
[3] S. Basu, New results on quantifier elimination over real closed fields and applications to constraint databases, J. ACM 46 (4) (1999) 537–555.
[4] R. Dechter, Constraint Processing, Morgan Kaufmann, San Mateo, CA, 2003.

[5] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[6] G. Hillebrand, P. Kanellakis, H. Mairson, Database query languages embedded in the typed lambda calculus, Inform. and Comput. 127 (2) (1996) 117–144.
[7] W. Hodges, A Shorter Model Theory, Cambridge University Press, Cambridge, 1997.
[8] P. Ladkin, Satisfying first-order constraints about time intervals, in: Proc. National Conference of the AAAI, 1989, pp. 512–517.
[9] C. Okasaki, Purely Functional Data Structures, Cambridge University Press, Cambridge, 1998.
[10] M. Scott, Programming Language Pragmatics, Morgan Kaufmann, San Mateo, CA, 2000.
[11] C. Umans, The minimum equivalent DNF problem and shortest implicants, J. Comput. System Sci. 63 (2001) 597–611.
[12] H. Wilf, Generatingfunctionology, second ed., Academic Press, New York, 1994.