

Query Rewriting with Symmetric Constraints^{*}

Christoph Koch

Database and Artificial Intelligence Group
Technische Universität Wien,
A-1040 Vienna, Austria
koch@dbai.tuwien.ac.at

We address the problem of answering queries using expressive *symmetric* inter-schema constraints which allow to establish mappings between several heterogeneous information systems. This problem is of high relevance to data integration, as symmetric constraints are essential for dealing with true concept mismatch and are generalizations of the kinds of mappings supported by both local-as-view and global-as-view approaches that were previously studied in the literature. Moreover, the flexibility gained by using such constraints for data integration is essential for virtual enterprise and e-commerce applications. We first discuss resolution-based methods for computing maximally contained rewritings and characterize computability aspects. Then we propose an alternative but semantically equivalent perspective based on a generalization of results relating to the database-theoretic problem of answering queries using views. This leads to a fast query rewriting algorithm based on AI techniques, which has been implemented and experimentally evaluated.

1. Introduction

This article addresses the *query rewriting problem* in semantic data integration in a very general form, as a proper generalization of the well-known local-as-view (e.g., [20,12,3]) and global-as-view approaches (e.g., [11,2,6]).

We focus on the relational case. Given a conjunctive query Q , we attempt to find a *maximally contained rewriting* in terms of a set of distinguished *source predicates* \mathbf{S} only – under a given

set of constraints, the positive relational queries as the output query language (i.e., a rewriting is a set of conjunctive queries), and the classical logical semantics. Under the classical semantics, (1) for each conjunctive query Q' in the maximally contained rewriting of Q , the constraints taken as a logical theory imply $Q \supseteq Q'$ and Q' uses predicates of \mathbf{S} only, and (2) for each conjunctive query Q'' over predicates in \mathbf{S} only for which the constraints imply that $Q \supseteq Q''$, there is a conjunctive query Q' in the rewriting such that the constraints imply $Q' \supseteq Q''$.

We support inter-schema constraints in the form of what we call *Conjunctive Inclusion Dependencies* (cind's), containment relationships between conjunctive queries. We refer to cind's as *symmetric* because they are syntactically symmetric with respect to the inclusion resp. implication symbol, while for instance materialized view definitions used for local-as-view query rewriting are not.

Example 1.1 Consider a conjunctive query

$$Q(x_1) \leftarrow \text{parent}(x_1, x_2) \wedge \text{parent}(x_2, x_3) \wedge \\ \text{parent}(x_3, x_4).$$

asking for great-grandparents in terms of a schema which contains a predicate “parent”. Now let “parent” be a “logical” relation to which database relations first need to be mapped before any queries over it can be answered. Let “parent” be conceived to represent parent-child relationships between living persons only. We want to rewrite Q into a query over source predicates in $\mathbf{S} = \{\text{grandparent}, \text{alive}\}$, where “grandparent” contains grandparent relationships between persons of which the grandparents may possibly have deceased and the source relation “alive” holds persons still alive. We may assert the cind

$$\{\langle x, z \rangle \mid \exists y : \text{parent}(x, y) \wedge \text{parent}(y, z)\} \supseteq \\ \{\langle x, z \rangle \mid \text{grandparent}(x, z) \wedge \text{alive}(x)\}$$

^{*} A short version of this article has appeared in *Proc. 2nd Int. Symp. on Foundations of Information and Knowledge Systems*, 2002 [16].

between the schema of “parent” and \mathbf{S} , i.e., a mapping formulated as a containment relationship between conjunctive queries. Then,

$$Q'(x_1) \leftarrow \text{grandparent}(x_1, x_3) \wedge \text{alive}(x_1) \wedge \text{grandparent}(x_3, z) \wedge \text{alive}(x_3).$$

(i.e., a *single* conjunctive query) is the maximally contained rewriting of Q , i.e. the largest query logically contained in Q with respect to the constraint given (and thus cannot return any wrong answers) that can be computed by only considering query and constraint, but not the data in the relations of \mathbf{S} , and which only uses relations from \mathbf{S} . \square

Note that the problem we attack in this article is different from work on query answering using integrity constraints [9,10,13] or the problem of optimizing queries in mediated systems. Our constraints are meant to encode mappings between the schemata of several information systems as in [5,4]; as such, they compare to materialized view *definitions* in the local-as-view approach rather than to classical *integrity constraints*.

cind’s may represent mappings between complex networks of information systems in which each information system may have its own schema and make use of integrated data. We are thus not restricted to a single layer of constraints (which map sources against a so-called “global” schema) as in the local-as-view approach. Starting from a set of actual source relations which contain data, views or “logical relations” can be defined and used to extend any of the schemata. To facilitate practical usability, a logical relation may map to sources through several indirections. Such an architecture is essential for large and open environments such as experienced in the modern e-economy [5,15].

The relevance of query rewriting with symmetric constraints stems from the observation that both local-as-view and global-as-view approaches to data integration are unable to deal with concept mismatch requiring mappings between pairs of complex query expressions. This problem is particularly common in complex technical domains [15].

Example 1.2 Assume that we have a number of sources holding information about computer models as-built (in fact, we only consider computer mainboards in this short example), their constituent parts, and technical specifications. We want to integrate these sources against a reference

design view \mathbf{R} with predicates “mb” (for mainboards), “cpu”, “cache”, “conn” (for connections or part-of relationships between component parts), and possibly others. Let a source schema \mathbf{S} represent mainboards using a CPU model called “p1” which has an on-chip cache. We cannot directly map cache components in this example, but we can map characteristics represented by both schemata (say, MHz rates r and cache sizes s) to mainboards and CPUs and mediate useful queries over \mathbf{R} . We encode the desired mapping using the cind

$$\begin{aligned} \{ \langle x, y, r, s \rangle \mid \exists z : \mathbf{R}.mb(x) \wedge \mathbf{R}.conn(x, y) \wedge \\ \mathbf{R}.conn(x, z) \wedge \mathbf{R}.cpu(y, r) \wedge \\ \mathbf{R}.cache(z, s) \} \supseteq \\ \{ \langle x, y, r, s \rangle \mid \mathbf{S}.mb(x) \wedge \mathbf{S}.conn(x, y) \wedge \\ \mathbf{S}.p1(y, r, s) \} \quad \square \end{aligned}$$

Local-as-view and global-as-view approaches assume that mediated schemata can be designed beforehand using intuitions concerning the likely sources to be added to an integration system later. Both large data integration settings and changing requirements render such an assumption unsustainable (see [15]). Our approach allows to “patch” local-as-view or global-as-view integration systems when sources need to be integrated whose particularities have not been foreseen when designing the schemata against which data are to be integrated.

The only previous work dealing with symmetric constraints is the description logics approach to data integration (e.g., [4,5]), which, however, requires high-complexity reasoning over the data (thus, there is a scalability issue) and to import all data to be integrated into the description logics reasoner. This is usually not an option in open e-economy or WWW data integration environments. Solving the integration problem on the level of queries and mappings *only* is essential for being able to deal with large amounts of data and restricted (e.g., screen-scraping) query interfaces.

Contributions and Structure. After some preliminaries (Section 2), we discuss a simple resolution-based method for generating rewritings and provide characterizations of the main theoretical properties of our problem in Section 3. Unfortunately, positive (and thus non-recursive) maximally contained rewritings may be infinite and the major decision problems (such as the non-emptiness or boundedness of the result) are undecidable. However, given that the predicate depen-

dependency graph (with respect to the inclusion direction) of a set of constraints is acyclic, we can guarantee to find the maximally contained rewritings, which are finite. The acyclic case is a proper generalization of both local-as-view and global-as-view approaches.

In Section 4, we propose an alternative algorithm for computing maximally contained rewritings which is based on a generalization of the Mini-Con Algorithm [22] for the problem of answering queries using views, and demonstrate its soundness and completeness. When using this algorithm, all intermediate rewritings are guaranteed to be function-free and thus conjunctive queries. Because of that, one can make use of classical database techniques for optimizing the rewriting process. Section 5 presents refinements of the algorithm of Section 4, which we have implemented in a practical system.

We evaluate our implementation, which is publicly available, experimentally in Section 6. It turns out that it scales to thousands of constraints and realistic applications. Section 7 concludes with a discussion of our new algorithm.

2. Preliminaries

We define a *conjunctive inclusion dependency* (cind) as a constraint of the form $Q_1 \subseteq Q_2$ where Q_1 and Q_2 are conjunctive queries of the form

$$\{\langle x_1, \dots, x_n \rangle \mid \exists x_{n+1} \dots x_m : (p_1(\bar{X}_1) \wedge \dots \wedge p_k(\bar{X}_k))\}$$

with *distinct*¹ unbound variables x_1, \dots, x_n , without arithmetic comparisons, but possibly with constants. We may write $\{Q_1 \equiv Q_2\}$ as a short form of $\{Q_1 \subseteq Q_2, Q_1 \supseteq Q_2\}$.

The semantics of cind's is the classical logical one and will be formally introduced in the framework of query rewriting in the next section. For some background in logic-based data integration and on GAV and LAV, we refer to the survey [18].

The *normal form* $NF(\Sigma)$ of a set Σ of cind's – i.e., Σ taken as a logical formula transformed into (implication) normal form – is a set of Horn clauses of a simple pattern. Every cind σ of the form $Q_1 \subseteq Q_2$ with

$$\begin{aligned} Q_1 &= \{\langle x_1, \dots, x_n \rangle \mid \exists x_{n+1} \dots x_m : \\ &\quad v_1(\bar{X}_1) \wedge \dots \wedge v_k(\bar{X}_k)\} \\ Q_2 &= \{\langle y_1, \dots, y_n \rangle \mid \exists y_{n+1} \dots y_{m'} : \\ &\quad p_1(\bar{Y}_1) \wedge \dots \wedge p_{k'}(\bar{Y}_{k'})\} \end{aligned}$$

translates to k' Horn clauses $p_i(\bar{Z}_i) \leftarrow v_1(\bar{X}_1) \wedge \dots \wedge v_k(\bar{X}_k)$. where each $z_{i,j}$ of \bar{Z}_i is determined as follows: If $z_{i,j}$ is a variable y_h with $1 \leq h \leq n$, replace it with x_h . If $z_{i,j}$ is a variable y_h with $n < h \leq m'$, replace it with Skolem function $f_{\sigma, y_h}(x_1, \dots, x_n)$ (the subscript assures that the Skolem functions are unique for a given constraint and variable).

Example 2.1 Let σ be the cind

$$\{\langle y_1, y_2 \rangle \mid \exists y_3 : p_1(y_1, y_3) \wedge p_2(y_3, y_2)\} \supseteq \{\langle x_1, x_2 \rangle \mid \exists x_3 : v_1(x_1, x_2) \wedge v_2(x_1, x_3)\}.$$

Then, $NF(\{\sigma\})$ is

$$\begin{aligned} p_1(x_1, f_{\sigma, y_3}(x_1, x_2)) &\leftarrow v_1(x_1, x_2) \wedge v_2(x_1, x_3). \\ p_2(f_{\sigma, y_3}(x_1, x_2), x_2) &\leftarrow v_1(x_1, x_2) \wedge v_2(x_1, x_3). \quad \square \end{aligned}$$

Whenever a cind translates into a function-free clause in normal form, we write it in datalog notation. This is the case for cind's $\{\langle \bar{X} \rangle \mid p(\bar{X})\} \supseteq Q$, i.e., where the subsumer queries are \exists -free single-literal queries.

The dependency graph of a set C of Horn clauses is the directed graph constructed by taking the predicates of C as nodes and adding, for each clause in C , an edge from each of the body predicates to the head predicate. The diameter of a directed acyclic graph is the longest directed path occurring in it. The dependency graph of a set of cind's Σ is the dependency graph of the logic program $NF(\Sigma)$. A set of cind's is *cyclic* if its dependency graph is cyclic. An acyclic set Σ of cind's is called *layered* if the predicates appearing in Σ can be partitioned into n disjoint sets P_1, \dots, P_n such that there is an index i for each cind $\sigma : Q_1 \subseteq Q_2 \in \Sigma$ such that $Preds(\text{Body}(Q_1)) \subseteq P_i$ and $Preds(\text{Body}(Q_2)) \subseteq P_{i+1}$ and $\mathbf{S} = P_1$.

3. Query Containment and Rewriting

Let us begin with a straightforward remark on the containment problem for conjunctive queries under a set of cind's Σ , which, since they are themselves containment relationships between conjunctive queries, is the implication problem for this type of constraint. To check a containment

¹Note that if we would not require unbound variables in constituent queries to be distinct, the transformation into normal form would result in Horn clauses with equality atoms as heads.

$$\{\langle \bar{X} \rangle \mid \exists \bar{Y} : \phi(\bar{X}, \bar{Y})\} \supseteq \{\langle \bar{X} \rangle \mid \exists \bar{Z} : \psi(\bar{X}, \bar{Z})\}$$

of two conjunctive queries under Σ by refutation (without loss of generality, we assume \bar{Y} and \bar{Z} to be disjoint and the unbound variables in the two queries above to be the same, \bar{X}), we have to show

$$\Sigma, \neg(\forall \bar{X} : (\exists \bar{Y} : \phi(\bar{X}, \bar{Y})) \leftarrow (\exists \bar{Z} : \psi(\bar{X}, \bar{Z}))) \models \perp$$

i.e. the inconsistency of the constraints and the negation of the containment taken together. In normal form, ψ becomes a set of ground facts where all variables have been replaced *one-to-one* by new constants and ϕ becomes a clause with an empty head, where all distinguished variables x_i have been replaced by constants also used in ψ .

Example 3.1 For proving

$$\Sigma \models \{\langle x_1, x_2 \rangle \mid \exists x_3 : p_1(x_1, x_3) \wedge p_2(x_3, x_2)\} \supseteq \{\langle y_1, y_2 \rangle \mid \exists y_3 : r_1(y_1, y_3) \wedge r_2(y_3, y_2)\}$$

for a set of cind's Σ , we create the logic program

$$\mathcal{P} := NF(\Sigma) \cup \{ \leftarrow p_1(\alpha_1, x_3) \wedge p_2(x_3, \alpha_2), \\ r_1(\alpha_1, \alpha_3) \leftarrow . \\ r_2(\alpha_3, \alpha_2) \leftarrow . \}$$

where $\alpha_1, \alpha_2, \alpha_3$ are constants not appearing elsewhere. By the correctness of resolution for logic programs, the containment above holds iff there is a refutation of the goal $\leftarrow p_1(\alpha_1, x_3) \wedge p_2(x_3, \alpha_2)$. with the remaining clauses in \mathcal{P} . \square

Definition 3.2 A set of conjunctive queries \mathcal{Q} is a maximally contained positive rewriting of a conjunctive query Q with respect to a set of cind's Σ and a set of source predicates \mathbf{S} iff

1. for each $Q' \in \mathcal{Q}$, $\Sigma \models Q \supseteq Q'$ and $Preds(Q') \subseteq \mathbf{S}$ and
2. for each conjunctive query Q'' with $\Sigma \models Q \supseteq Q''$ and $Preds(Q'') \subseteq \mathbf{S}$, there is a $Q' \in \mathcal{Q}$ such that $\Sigma \models Q \supseteq Q' \supseteq Q''$. \square

In the finite case, a minimal² such set \mathcal{Q} is of course unique up to reordering and variable renaming.

For simplicity, we assume that no source predicates appear in any heads of Horn clauses in $NF(\Sigma)$ throughout this article. This does not

cause any loss of generality, since we can always replace a source predicate that violates this assumption by a new virtual predicate in all cind's and then add a cind that maps the source predicate to that new virtual predicate.

Informally, we can obtain such a maximally contained rewriting by a method based on SLD resolution in the following way. Given a conjunctive query Q , a set of cind's Σ , and a set of source predicates \mathbf{S} , we first create a logic program $NF(\Sigma)$ and add a unit clause $s(\bar{X}) \leftarrow .$ (with a tuple \bar{X} of *distinct* variables) for each predicate $s \in \mathbf{S}$. Then we try to refute the body of Q . (Differently from what we do for containment, we do not freeze any variables.) If we have found a refutation with a most general unifier θ , we collect the unit clauses used and create a Horn clause with $\theta(Head(Q))$ as head and the application of θ to the copies of unit clauses involved in the proof as body. If this clause is function-free, we output it. After that *we go on* as if we had not found a “proof” to compute more rewritings. Given an appropriate selection rule or a breath-first strategy for computing derivations, it is easy to see that this method will compute a maximally contained rewriting of Q in terms of multi-sets of conjunctive queries in the sense that for each conjunctive query Q'' contained in Q , a subsumer Q' will eventually be produced s.t. $\Sigma \models Q \supseteq Q' \supseteq Q''$. See Example 4.9 for query rewriting by an altered refutation proof.

Computability and Complexity.

Theorem 3.3 *Let Σ be a set of cind's, \mathbf{S} a set of predicates, and Q and Q' be conjunctive queries. Then the following problems are undecidable:*

1. $\Sigma \models Q \subseteq Q'$, *the containment problem.*
2. $\exists Q' : \Sigma \models Q \supseteq Q'$ *s.t. $Preds(Q') \subseteq \mathbf{S}$, i.e. it is undecidable whether the maximally contained rewriting of a conjunctive query Q w.r.t. Σ and \mathbf{S} is nonempty (that is, it contains at least one conjunctive query).*

Moreover, the boundedness problem (cf. [1]) for maximally contained positive rewritings is undecidable, as any datalog program can be written as a set of cind's.

Post's Correspondence Problem (PCP, see e.g. [24]), a simple and well-known undecidable problem, is defined as follows. Given nonempty words x_1, \dots, x_n and y_1, \dots, y_n over the alphabet $\{0, 1\}$,

²A set of conjunctive queries is minimal if and only if the constituent queries are individually minimal and pairwise non-redundant.

the problem is to decide whether there are indexes i_1, \dots, i_k (with $k > 0$) s.t. $x_{i_1}x_{i_2}\dots x_{i_k} = y_{i_1}y_{i_2}\dots y_{i_k}$. Pairs of words $\langle x_i, y_i \rangle$ are also called *dominos*.

In the following example, we provide a reduction of PCP to our query rewriting problem that proves the undecidability results of Theorem 3.3.

Example 3.4 Let s be a source, $q \leftarrow \text{inc!}(0, 0)$, a boolean query, and

$$\begin{aligned} \text{inc!}(x, y) \leftarrow & \text{one}(x, x_1) \wedge \text{zero}(x_1, x_2) \wedge \\ & \text{one}(x_2, x_3) \wedge \text{one}(y, y_1) \wedge \\ & \text{inc}(x_3, y_1). \end{aligned} \quad (1)$$

$$\begin{aligned} \text{inc!}(x, y) \leftarrow & \text{one}(x, x_1) \wedge \text{zero}(y, y_1) \wedge \\ & \text{one}(y_1, y_2) \wedge \text{one}(y_2, y_3) \wedge \\ & \text{one}(y_3, y_4) \wedge \text{zero}(y_4, y_5) \wedge \\ & \text{inc}(x_1, y_5). \end{aligned} \quad (2)$$

$$\text{inc}(x, y) \leftarrow \text{inc!}(x, y). \quad (3)$$

$$\text{inc}(x, y) \leftarrow \text{dec}(x, y). \quad (4)$$

$$\begin{aligned} \{\langle x, y \rangle \mid \text{dec}(x, y)\} \subseteq \\ \{\langle x, y \rangle \mid \exists x_1, y_1 : \text{zero}(x, x_1) \wedge \text{zero}(y, y_1) \wedge \\ \text{dec}(x_1, y_1)\} \end{aligned} \quad (5)$$

$$\begin{aligned} \{\langle x, y \rangle \mid \text{dec}(x, y)\} \subseteq \\ \{\langle x, y \rangle \mid \exists x_1, y_1 : \text{one}(x, x_1) \wedge \text{one}(y, y_1) \wedge \\ \text{dec}(x_1, y_1)\} \end{aligned} \quad (6)$$

$$\text{dec}(0, 0) \leftarrow s. \quad (7)$$

seven cind's of which the leading two stand for the instance

$$\mathcal{I} = \{\langle x_1 = 101, y_1 = 1 \rangle, \langle x_2 = 1, y_2 = 01110 \rangle\}$$

and the remaining four constitute the core PCP encoding. The constraints (1) and (2) “guess” two words represented as chains of “one” and “zero” atoms by the nondeterminism by which resolution (or MCD rewriting using Algorithm 4.3, for that matter) chooses a clause to resolve an “inc” atom, (3) and (4) allow for a choice whether further dominos are to be appended or not, constraints (5) and (6) “check” whether the two words are equal (which indicates the existence of a solution to the PCP problem) by proceeding from the right to the left, and constraint (7) “terminates” if the search was successful.

For showing the PCP instance \mathcal{I} satisfiable, one can compute a contained rewriting by applying the constraints in the following order (we only describe the proof but no dead-end branches): (guess phase)

(1), (3), (2), (3), (1), (4), (check phase) (6), (5), (6), (6), (6), (5), (6), (termination) (7).³

We find a solution $x_1x_2x_1 = y_1y_2y_1 = 1011101$ to \mathcal{I} . Generally, a PCP instance is satisfiable iff the maximally contained rewriting is $\{q \leftarrow s\}$. (Furthermore, a PCP instance is satisfiable iff $\Sigma \models \{\langle \rangle \mid \text{inc}(0, 0)\} \supseteq \{\langle \rangle \mid s\}$.) \square

For the important case that Σ is acyclic, the above problems are decidable (and those of Theorem 3.3 are NEXPTIME-complete). We first establish the following auxiliary result.

Lemma 3.5 Let Σ be an acyclic set of cind's and Q and Q' be conjunctive queries. Then the containment problem $\Sigma \models Q \subseteq Q'$ and the problem of deciding whether the maximally contained rewriting of Q (as a set of conjunctive queries) is nonempty are NEXPTIME-hard. \square

Proof. NEXPTIME-hardness follows from a slightly altered form of the encoding of the NEXPTIME-complete Tiling problem (see e.g. [21]) used in [7] to show NEXPTIME-hardness of the SUCCESS problem for nonrecursive logic programming, i.e., the problem of deciding whether a nonrecursive logic program treated as a database query will return a nonempty result.

TILING is the problem of tiling the square of size $2^n \times 2^n$ by tiles (squares of size 1×1) of k types. There are two binary relations **on** and **to** defined on the tiles. Tiles t_i and t_j are said to be *horizontally compatible* if $\langle t_i, t_j \rangle \in \mathbf{to}$ holds and are called *vertically compatible* if $\langle t_i, t_j \rangle \in \mathbf{on}$. A tiling of the square of size $2^n \times 2^n$ is a function $f : \{1, \dots, 2^n\} \times \{1, \dots, 2^n\} \rightarrow \{t_1, \dots, t_k\}$ such that vertically and horizontally neighboring tiles are compatible, i.e. $\langle f(i, j), f(i+1, j) \rangle \in \mathbf{to}$ for all $1 \leq i < 2^n, 1 \leq j \leq 2^n$ and $\langle f(i, j), f(i, j+1) \rangle \in \mathbf{on}$ for all $1 \leq i \leq 2^n, 1 \leq j < 2^n$.

The TILING problem is defined as follows. Suppose that we are given a set $\{t_1, \dots, t_k\}$ of tiles, compatibility relations **on** and **to**, and a number n written in unary notation, the problem is to decide whether there exists a tiling f of the square of size $2^n \times 2^n$ with a distinguished tile type, say t_1 , at the top left corner (i.e., $f(1, 1) = t_1$).

³One can easily verify this using the intuition of fully replacing parts of (intermediate) goals by subsumed queries of cind's whose subsumer queries fully match those parts. Due to the special structure of the cind's in this reduction, at any point, all MCDs – to be introduced in Definition 4.1 – are “isomorphic” to some subsumer query of a cind.

x_1	x_2	y_1	y_2
x_3	x_4	y_3	y_4
z_1	z_2	u_1	u_2
z_3	z_4	u_3	u_4

x_1	x_2	x_2	y_1	y_1	y_2
x_3	x_4	x_4	y_3	y_3	y_4
x_3	x_4	x_4	y_3	y_3	y_4
z_1	z_2	z_2	u_1	u_1	u_2
z_1	z_2	z_2	u_1	u_1	u_2
z_3	z_4	x_4	u_3	u_3	u_4

Fig. 1. Hypertile of size $i \geq 2$ (left) and all nine possible overlapping hypertiles of size $i - 1$ that can be inscribed into it (right).

We describe a reduction that transforms any instance of the tiling problem to an instance of the containment problem of conjunctive queries under an acyclic set of cind's and which requires only polynomial time relative to the size of the problem instance.

We define *hypertiles* as follows. Each composition of 2×2 tiles or hypertiles is a hypertile if the component tiles satisfy the compatibility constraints. Obviously, all hypertiles are of size $2^i \times 2^i$ for some $i \geq 1$ [7].

In our encoding, we use atoms of the form

$$\text{til}_i(f, x, y, z, u, t)$$

to assert that hypertile f of level i is composed of the four hypertiles $x, y, z,$ and u of level $i - 1$ s.t. x is the left-top, y the right-top, z the left-bottom, and u the right-bottom constituent hypertile, and that t is the left- and topmost *tile* in f . (Of course t must be equal to or form part of x .)

We define hypertiles of level 1 by the cind

$$\begin{aligned} & \{ \langle x_1, x_2, x_3, x_4 \rangle \mid \exists x_f : \text{til}_1(x_f, x_1, x_2, x_3, x_4, x_1) \} \supseteq \\ & \{ \langle x_1, x_2, x_3, x_4 \rangle \mid \text{to}(x_1, x_2) \wedge \text{to}(x_3, x_4) \wedge \\ & \quad \text{on}(x_1, x_3) \wedge \text{on}(x_2, x_4) \} \end{aligned}$$

Fortunately, for hypertiles of level $i \geq 2$, it is not necessary to enforce that all the compatibility constraints are satisfied on the level of tiles. Instead, it is sufficient to verify that all of the nine possible (overlapping) constituent hypertiles of the next-smaller level $i - 1$ (see Figure 1) satisfy the compatibility constraints. We define hypertiles of level greater than one by

$$\begin{aligned} & \{ \langle x_f, y_f, z_f, u_f, t \rangle \mid \exists f : \text{til}_{i+1}(f, x_f, y_f, z_f, u_f, t) \} \supseteq \\ & \{ \langle x_f, y_f, z_f, u_f, t \rangle \mid \exists x_1, \dots, x_4, y_1, \dots, y_4, \\ & \quad z_1, \dots, z_4, u_1, \dots, u_4, \\ & \quad d_1, \dots, d_{13} : \\ & \quad \text{til}_i(x_f, x_1, x_2, x_3, x_4, t) \wedge \\ & \quad \text{til}_i(y_f, y_1, y_2, y_3, y_4, d_1) \wedge \end{aligned}$$

$$\begin{aligned} & \text{til}_i(z_f, z_1, z_2, z_3, z_4, d_2) \wedge \\ & \text{til}_i(u_f, u_1, u_2, u_3, u_4, d_3) \wedge \\ & \text{til}_i(d_4, x_2, y_1, x_4, y_3, d_5) \wedge \\ & \text{til}_i(d_6, x_4, y_3, z_2, u_1, d_7) \wedge \\ & \text{til}_i(d_8, z_2, u_1, z_4, u_3, d_9) \wedge \\ & \text{til}_i(d_{10}, x_3, x_4, z_1, z_2, d_{11}) \wedge \\ & \text{til}_i(d_{12}, y_3, y_4, u_1, u_2, d_{13}) \} \end{aligned}$$

Let bot be a nullary predicate. To complete our encoding, we add cind's $\text{on}(t_i, t_j) \leftarrow \text{bot}$. for each $\langle t_i, t_j \rangle \in \mathbf{on}$ and $\text{to}(t_i, t_j) \leftarrow \text{bot}$. for each $\langle t_i, t_j \rangle \in \mathbf{to}$. where t_i and t_j are constants identifying pairs out of the k given tile types.

Let us consider the encoding shown above as a logic program (that we obtain by normalizing the cind's). The existential variables in the subsumer queries of the til_i cind's will be transformed into function terms aggregating the 4 hypertiles of the next smaller size. (In fact, also the variables for the top left corner tiles t will be aggregated in the function terms, but this does not alter the correctness of the encoding.) The cind for til_1 is transformed into the Horn clause

$$\begin{aligned} & \text{til}_1(f_1(x_1, x_2, x_3, x_4), x_1, x_2, x_3, x_4, x_1) \leftarrow \\ & \quad \text{to}(x_1, x_2), \text{to}(x_3, x_4), \text{on}(x_1, x_3), \text{on}(x_2, x_4). \end{aligned}$$

and the cind's for $\text{til}_{i \geq 2}$ are normalized as Horn clauses with heads

$$\text{til}_i(f_i(x_1, x_2, x_3, x_4, t), x_1, x_2, x_3, x_4, t)$$

During bottom-up evaluation of such a logic program, the function terms constructed using f_i correspond exactly with the valid hypertiles constructible from the given k tile types, if the fifth arguments of function terms of symbols $f_{i \geq 2}$ are ignored.

It is easy to see that there is a solution for the TILING problem iff the constraints in our encoding entail $\{ \langle \rangle \mid \text{bot} \} \subseteq \{ \langle \rangle \mid \text{til}_m(f, x, y, z, u, 1) \}$. Equally, there is a solution to the TILING problem exactly if the maximally contained rewriting of $\{ \langle \rangle \mid \text{til}_m(f, x, y, z, u, 1) \}$ in terms of the ‘‘source predicate’’ bot is nonempty. Thus, these two problems are NEXPTIME-hard. \square

Theorem 3.6 *Let Σ be an acyclic set of cind's and Q and Q' be conjunctive queries. Then the containment problem $\Sigma \models Q \subseteq Q'$ and the query rewriting problem for conjunctive queries (under acyclic sets of cind's) are NEXPTIME-complete.*

Proof. As pointed out in Section 3, the query containment problem under an acyclic set of cind's can be solved by proving the unsatisfiability of the negation of the containment, which decomposes into a set of ground facts and a goal. This is a special case of the SUCCESS problem for nonrecursive logic programs [7,25].

The problem of deciding whether query rewriting produces a nonempty set of conjunctive queries can be reduced to the SUCCESS problem by introducing unit clauses $s_i(x_1, \dots, x_{n_i}) \leftarrow$. (where x_1, \dots, x_{n_i} are distinct variables) for each "source" predicate s_i of arity n_i .

As both problems are known NEXPTIME-hard from Lemma 3.5, completeness in NEXPTIME has been shown. \square

4. Generalizing Local-as-view Rewriting

The results of this section generalize from work on algorithms for the problem of answering queries using views [19], for instance the Bucket Algorithm [20], the Inverse Rules Algorithm [9], OCCAM [17], the Unification-join Algorithm [23], and particularly the MiniCon Algorithm [22]. For space reasons, we introduce necessary notions as needed and refer to [22] for a discussion and comparison of such algorithms.

We adapt the notion of MiniCon descriptions [22] to our framework based on query rewriting with cind's decomposed into Horn clauses.

Definition 4.1 (Inverse MiniCon Description). Let Q be a conjunctive query with $n = |Body(Q)|$ and Σ be a set of cind's. An (inverse) MiniCon description (MCD) for Q is a pair of a tuple $\langle c_1, \dots, c_n \rangle \in (NF(\Sigma) \cup \{\epsilon\})^n$ with at least one $c_i \neq \epsilon$ and a substitution θ that satisfies the following two conditions.

1. For the most general unifier $\theta \neq fail$ arrived at by unifying the heads of all the $c_i \neq \epsilon$ with $Body_i(Q)$, the unfolding of Q and $\langle c_1, \dots, c_n \rangle$ under θ is function-free and
2. there is no tuple

$$\langle c'_1, \dots, c'_n \rangle \in \{c_1, \epsilon\} \times \dots \times \{c_n, \epsilon\}$$

with fewer entries different from ϵ than in $\langle c_1, \dots, c_n \rangle$, such that the unfolding of Q with $\langle c'_1, \dots, c'_n \rangle$ is function free. \square

Example 4.2 Consider again the query Q and the constraint (which we now call σ) of Example 1.1. $NF(\{\sigma\})$ is

$$c_1 : \text{parent}(x, f_{\sigma,y}(x, z)) \leftarrow \text{grandparent}(x, z) \wedge \text{alive}(x).$$

$$c_2 : \text{parent}(f_{\sigma,y}(x, z), z) \leftarrow \text{grandparent}(x, z) \wedge \text{alive}(x).$$

We have two MCDs, $\langle \langle c_1, c_2, \epsilon \rangle, \theta \rangle$ with the unifier

$$\theta = \{[x_1/x^{(1)}], [x_2/f_{\sigma,y}(x^{(1)}, z^{(1)})], [x_2/f_{\sigma,y}(x^{(2)}, z^{(2)})], [x_3/z^{(2)}]\}$$

and $\langle \langle \epsilon, \epsilon, c_1 \rangle, \theta' \rangle$ with

$$\theta' = \{[x_3/x^{(3)}], [x_4/f_{\sigma,y}(x^{(3)}, z^{(3)})]\}.$$

Note that $\langle c_1, c_2, c_1 \rangle$ violates condition 2 of Definition 4.1, while all other MCD candidates violate condition 1. \square

Note that the inverse MiniCon descriptions of Definition 4.1 exactly coincide with the MCDs of [22] in the local-as-view case. Algorithm 4.3 shown below can easily be reformulated so as to use a slight generalization of the notation of [22] to cover clause bodies consisting of several atoms. That way, one can even escape the need to transform cind's into Horn clauses and can reason completely without the introduction of function terms. However, to support the presentation of our results (particularly the equivalence proof of the following section), we do not follow this path in this article.

Algorithm 4.3 (Query rewriting with MCDs).

Input. A conjunctive query Q , a set of cind's Σ , and a set \mathbf{S} of source predicates

Output. A maximally contained rewriting of Q w.r.t. Σ and \mathbf{S}

```

 $Qs := [Q];$ 
while  $Qs$  is not empty do
{
   $[Q, Qs] := Qs;$ 
  if  $Preds(Q) \subseteq \mathbf{S}$  then output  $Q;$ 
  else
  {
     $M :=$  compute the set of all inverse MCDs
      for  $Q$  and  $\Sigma;$ 
    for each  $\langle \langle c_1, \dots, c_n \rangle, \theta \rangle \in M$  do
    {
       $Q' :=$  unfold( $Q, \theta, \langle c_1, \dots, c_n \rangle$ );
       $Qs := [Qs, Q'];$ 
    }
  }
}

```

\square

In Algorithm 4.3, maximally contained rewritings of a conjunctive query Q are computed by iteratively unfolding queries with *single* MiniCon descriptions⁴ until a rewriting contains only source predicates in its body. In order to handle cyclic sets of cind’s (and attain completeness), we manage intermediate rewritings using a queue and, consequently, follow a breath-first strategy.

The function “unfold” accepts a conjunctive query Q with $|Body(Q)| = n$, a unifier θ and a tuple of n Horn clauses or ϵ s.t. if $c_i \neq \epsilon$, θ unifies $Body_i(Q)$ with $Head(c_i)$. It produces a new clause from Q (which in fact is again guaranteed to be function-free and thus a conjunctive query) by replacing $Head(Q)$ by $\theta(Head(Q))$ and each of the non-source body atoms $Body_i(Q)$, with $c_i \neq \epsilon$, by $\theta(Body(c_i))$. (i.e. after applying substitutions from the unifier). If $c_i = \epsilon$, $Body_i(Q)$ is replaced by $\theta(Body_i(Q))$. Of course, for each MCD $\langle \langle \dots, c_i, \dots, c_j, \dots \rangle, \theta \rangle$ we have $\theta(Body(c_i)) = \theta(Body(c_j))$, and thus only one rule body needs to be added for each MCD during unfolding.

Theorem 4.4 *Let Q be a conjunctive query, Σ be a set of cind’s, and \mathbf{S} be a set of “source” predicates. Then, for each conjunctive query Q' with $Preds(Q') \subseteq \mathbf{S}$ we have $\Sigma \models Q \supseteq Q'$ iff Algorithm 4.3 eventually computes a conjunctive query Q'' with $Preds(Q'') \subseteq \mathbf{S}$ and $\Sigma \models Q \supseteq Q'' \supseteq Q'$.*

In other words, Algorithm 4.3 enumerates the maximally contained positive rewriting of Q under Σ in terms of \mathbf{S} .

The notion of a *rewrite system* provides a strong intuition for the technique proposed by Algorithm 4.3 and Theorem 4.4:

Example 4.5 (“Coffee Can Problem” [8]) Consider the rewrite system

black white \rightarrow black

white black \rightarrow black

black black \rightarrow white

with symbols “white” and “black” and the input word $w = (\text{white white black black white white black black})$ where the goal is to replace sequences

of symbols of that word that match the left hand side of one of the three productions listed above repeatedly to produce a rewriting that is as small as possible. One such sequence of replacements is

- (0) white white black black white white black black
- (1) white white black black white black black
- (2) white white white white black black
- (3) white white white black black
- (4) white white black black
- (5) white black black
- (6) black black
- (7) white

Pairs of occurrences of the symbols “black” or “white” have been underlined immediately before their replacement. Thus, the input string can be rewritten into a word with a single symbol, “white”.

We can simulate such behavior using query rewriting under the rewrite systems semantics. Let us search for one-symbol rewritings. We model an n -symbol word $w \in \{\text{black}, \text{white}\}^n$ as a query of the form

$$q(x_1) \leftarrow \text{start_end}(x_1, x_{n+1}), p_1(x_1, x_2), \dots, \\ p_i(x_i, x_{i+1}), \dots, p_n(x_n, x_{n+1}).$$

where p_i is either “black” or “white”, equal to the i -th symbol of w , and $x_1 \dots x_{n+1}$ are variables. The above input word is thus represented as

$$q(x_1) \leftarrow \text{start_end}(x_1, x_9), \\ \text{white}(x_1, x_2), \text{white}(x_2, x_3), \text{black}(x_3, x_4), \\ \text{black}(x_4, x_5), \text{white}(x_5, x_6), \text{white}(x_6, x_7), \\ \text{black}(x_7, x_8), \text{black}(x_8, x_9).$$

The rewrite system can be encoded as a set of cind’s

$$\begin{aligned} \{ \langle x, y \rangle \mid \exists z : \text{black}(x, z) \wedge \text{white}(z, y) \} &\supseteq \\ \{ \langle x, y \rangle \mid \text{black}(x, y) \} & \\ \{ \langle x, y \rangle \mid \exists z : \text{white}(x, z) \wedge \text{black}(z, y) \} &\supseteq \\ \{ \langle x, y \rangle \mid \text{black}(x, y) \} & \quad (\star) \\ \{ \langle x, y \rangle \mid \exists z : \text{black}(x, z) \wedge \text{black}(z, y) \} &\supseteq \\ \{ \langle x, y \rangle \mid \text{white}(x, y) \} & \end{aligned}$$

Furthermore, we define two source predicates w_src and b_src and define cind’s responsible for making the rewrite process terminate with “success” (i.e., a contained rewriting in terms of the source predicates is found).

$$\begin{aligned} \{ \langle x \rangle \mid \exists y : \text{start_end}(x, y) \wedge \text{white}(x, y) \} &\supseteq \\ \{ \langle x \rangle \mid w_src(x) \} & \\ \{ \langle x \rangle \mid \exists y : \text{start_end}(x, y) \wedge \text{black}(x, y) \} &\supseteq \\ \{ \langle x \rangle \mid b_src(x) \} & \end{aligned}$$

⁴In this respect, Algorithm 4.3 differs from the MiniCon algorithm for the problem of answering queries using views, where MCDs are packed so as to rewrite all body atom at once.

It can be verified by applying Algorithm 4.3 (although this is a quite work-intensive task to do by hand) that the maximally contained rewriting under the rewrite systems semantics is $q'(x_1) \leftarrow \text{w_src}(x_1)$.

In fact, the seven-step sequence of replacements shown above can be easily used to create a proof in our rewrite systems semantics that q' is *in* the maximally contained rewriting. For the first replacement of that sequence, the tuple $\langle c_1, \dots, c_n \rangle \in (C \cup \{\epsilon\})^n$ of Algorithm 4.3 would equal $\langle \epsilon, \epsilon, \epsilon, c_{\sigma_2,1}, c_{\sigma_2,2}, \epsilon, \epsilon, \epsilon \rangle$ where $c_{\sigma_2,1}$ and $c_{\sigma_2,2}$ are the first and second Horn clause created by normalizing our second *cind* (\star). We can conclude that the above rewrite system cannot result in a one-symbol rewriting “black” for the given input word. \square

Correctness of Theorem 4.4.

It is easy to see that the rewriting process of Algorithm 4.3 simply is equivalent to resolution where only some of the subgoals of a goal may be rewritten in a single step and each intermediate rewriting has to be function-free. Every proof generated by Algorithm 4.3 is thus a correct resolution proof. Thus,

Lemma 4.6 (*Soundness of Algorithm 4.3*) *Let Q be a conjunctive query, Σ a set of *cind*'s, and \mathbf{S} a set of source predicates. Then, for each conjunctive query Q' generated by Algorithm 4.3 for Q , Σ , and \mathbf{S} , we have $\Sigma \models Q \supseteq Q'$ and $\text{Preds}(Q') \subseteq \mathbf{S}$.*

Completeness is a consequence of the following result.

Lemma 4.7 *Let \mathcal{P} be a resolution proof establishing a logically contained rewriting of a conjunctive query Q under a set of *cind*'s Σ . Then, there is always a proof \mathcal{P}' establishing the same contained rewriting such that each intermediate rewriting is function-free.*

Proof. Let us assume that each new subgoal a derived using resolution receives an identifying index $\text{idx}(a)$. Then, given the proof \mathcal{P} , there is a unique *next premise to be applied* $c_{\text{idx}(a)}$ out of the Horn clauses in $NF(\Sigma)$ for each subgoal a . This is the Horn clause from our constraints base that will be unfolded with a to resolve it in \mathcal{P} .

Note that the proof \mathcal{P} is fully described by some unique indexing $\text{idx}(a)$ of all subgoals a appearing

in the proof (while we do not need to know or remember the atoms themselves), the clauses $c_{\text{idx}(a)}$, and a specification of which indexes the subgoals in the bodies of these clauses are attributed with when they are unfolded with subgoals.

In our original proof \mathcal{P} , each subgoal a of a goal is rewritten with $c_{\text{idx}(a)}$ in each step, transforming g_0 , the body of Q and at the same time the initial goal, via g_1, \dots, g_{n-1} to g_n , the body of the resulting rewriting. We maintain the head of Q separately across resolution steps and require that variables in the head are not unified with function terms, but apply other unifications effected on the variables in the goals in parallel with the rewriting process. Already \mathcal{P} must assure *at any step* that no variable from the head of Q is unified with a function term, as otherwise no conjunctive query can result.

We know that resolution remains correct no matter in which order the next due resolution steps $c_{\text{idx}(a)}$ are applied to the subgoals, and that we even may unfold, given e.g. a goal with two atoms, the first goal and then a subgoal from the unfolding of that first goal (and may do that any finite number of times) before we unfold our second original subgoal.

Coming back to deriving a function-free proof starting from \mathcal{P} , all we now have to show is that at any intermediate step of a resolution proof with *cind*'s, a nonempty set of subgoals $X = \{a_{i_1}, \dots, a_{i_k}\} \subseteq g_i$ of the function-free intermediate goal g_i exists such that, when only these subgoals are unfolded with their next due premises to be applied $c_{\text{idx}(a_{i_1})}, \dots, c_{\text{idx}(a_{i_k})}$, the overall new goal g_{i+1} produced will be function-free⁵. The emphasis here lies on finding a *nonempty* such set X , as the empty set automatically satisfies this condition. If we can guarantee that such a nonempty set always exists until the function-free proof has been completed, our lemma is shown.

Let there be a dependency graph $G_{g_i} = \langle V, E \rangle$ for each (intermediate) goal g_i with the subgoals as vertices and a directed edge $\langle a, b \rangle \in E$ iff a contains a variable v that is unified with a function term $f(\bar{X})$ in $\text{Head}(c_{\text{idx}(a)})$ and v appears in b and is unified with a variable (rather than a function term with the same function symbol) in $\text{Head}(c_{\text{idx}(b)})$. (Intuitively, if there is an edge

⁵The correctness of the proof \mathcal{P} alone assures that the query head will be function-free as well.

$\langle a, b \rangle \in E$, then b must be resolved *before* a if a proof shall be obtained in which all intermediate goals are function-free.) As mentioned, query heads are guaranteed to remain function-free by the correctness of \mathcal{P} . For instance, the dependency graph of the goal

$$\leftarrow a(x)^{(0)} \wedge b(x, y)^{(1)} \wedge c(y, z)^{(2)} \wedge d(z, w)^{(3)}.$$

with

$$\begin{aligned} c_0 : a(x) &\leftarrow a'(x). & c_1 : b(f(x), x) &\leftarrow b'(x). \\ c_2 : c(x, x) &\leftarrow c'(x). & c_3 : d(g(x), x) &\leftarrow d'(x). \end{aligned}$$

would be $G = \{\langle 0, 1, 2, 3 \rangle, \{\langle 1, 0 \rangle, \langle 3, 2 \rangle\}\}$, i.e. the first subgoal must be resolved before the second and the third subgoal must be resolved before the fourth.

We can now show that such a dependency graph G is *always acyclic*. In fact, if it were not, \mathcal{P} could not be a valid proof, because unification would fail when trying to unify a variable in such a cycle with a function term that *contains that variable*. This is easy to see because each function term given our construction used for obtaining Horn clauses from cind's contains *all* variables appearing in that same (head) atom. Consider for instance

$$q(x) \leftarrow a(x, y), a(y, z), b(w, z), b(z, y).$$

$$\begin{aligned} \{\langle x, y \rangle \mid \exists z : a(x, z) \wedge a(z, y)\} &\supseteq \{\langle x, y \rangle \mid s(x, y)\} \\ \{\langle x, y \rangle \mid \exists z : b(x, z) \wedge b(z, y)\} &\supseteq \{\langle x, y \rangle \mid s(x, y)\} \end{aligned}$$

where s is a source. There is no rewriting under our two semantics, because the dependency graph of our above construction is cyclic already for our initial goal, the body of q .

However, since G is acyclic given a proof \mathcal{P} , we can unfold a nonempty set of atoms (those unreachable from other subgoals in graph G) with our intermediate goals until the proof has been completed. \square

As an immediate consequence of Lemma 4.7 (which assures that for each resolution proof \mathcal{P} showing $\Sigma \models Q \supseteq Q'$ we can produce an equivalent function-free proof \mathcal{P}' that will be covered by Algorithm 4.3), we have

Lemma 4.8 (*Completeness of Algorithm 4.3*) *If $\Sigma \models Q \supseteq Q'$ and $\text{Preds}(Q') \subseteq \mathbf{S}$, then Algorithm 4.3 computes a conjunctive query Q'' s.t. $\Sigma \models Q \supseteq Q'' \supseteq Q'$.*

Lemma 4.6 and Lemma 4.8 taken together imply Theorem 4.4. Let us visualize the implications of Lemma 4.7 with an example.

Example 4.9 Given a boolean conjunctive query $q \leftarrow b(x, x, 0)$. and the following set of Horn clauses which, as is easy to see, are the normal form of a set of cind's, which we do not show in order to reduce redundancy.

$$\begin{aligned} b(x', y', 0) &\leftarrow a(x, y, 2) \wedge e_\epsilon(x, x') \wedge e_1(y, y'). & c_0 \\ b(x', y', 2) &\leftarrow a(x, y, 0) \wedge e_1(x, x') \wedge e_0(y, y'). & c_4, c_{10}, c_{11} \\ b(x', y', 0) &\leftarrow a(x, y, 1) \wedge e_0(x, x') \wedge e_\epsilon(y, y'). & c_{12}, c_{18}, c_{19} \\ b(x', y', 1) &\leftarrow a(x, y, 0) \wedge e_1(x, x') \wedge e_1(y, y'). & c_{20}, c_{25} \end{aligned}$$

$$\begin{aligned} e_\epsilon(x, x) &\leftarrow v(x). & c_2, c_{17} \\ e_1(x, f_1(x)) &\leftarrow v(x). & c_3, c_8, c_{23}, c_{24} \\ e_0(x, f_0(x)) &\leftarrow v(x). & c_2, c_{17} \\ v(x) &\leftarrow b(x, y, s). & c_5, c_{13}, c_{21} \\ v(y) &\leftarrow b(x, y, s). & c_6, c_{14} \\ a(x, y, s) &\leftarrow b(x, y, s). & c_1, c_7, c_{15} \end{aligned}$$

where x, y, x', y', s are variables. Let \mathcal{P} be the resolution proof in Figure 2 (a) which rewrites our query into $q \leftarrow a(x, x, 0), v(x)$. and in which we have superscribed each subgoal with its assigned index. In each resolution step, a goal $\leftarrow A^{(i_1)}, \dots, A^{(i_n)}$ is unfolded with the clauses c_{i_1}, \dots, c_{i_n} , as annotated above. To keep things short, we have eliminated subgoals (marked with a dagger \dagger and their index) that are redundant with a different branch of the proof. As claimed in our theorem, \mathcal{P} can be transformed into the proof in Figure 2 (b) in which each intermediate step is function-free. The subgoals that we have marked with brackets [] had been blocked at a certain step to keep the proof function-free. \square

Note that Example 4.9 constitutes another encoding of PCP that shows the undecidability of query rewriting with cind's. The PCP instance

$$\mathcal{I} = \{\langle x_1 = 10, y_1 = 1 \rangle, \langle x_2 = 1, y_2 = 01 \rangle\}$$

is encoded in the first four Horn clauses, which can be viewed as realizing a nondeterministic automaton that accepts two words $x_{i_1} \dots x_{i_k}$ and $y_{i_1} \dots y_{i_k}$ if they can be constructed using the dominos of \mathcal{I} . In the start state s_0 , a domino $\langle x_i, y_i \rangle$ out of \mathcal{I} is chosen. The symbols in x_i and y_i are then accepted one by one. If one of the two words x_i, y_i is longer than the other one, the shorter one is appended ϵ symbols. We return to

$$\begin{aligned}
(0) & \leftarrow b(x, x, 0)^{(0)}. \\
(1) & \leftarrow a(x, y, 2)^{(1)} \wedge e_\epsilon(x, z)^{(2)} \wedge e_1(y, z)^{(3)}. \\
(2) & \leftarrow b(f_1(y), y, 2)^{(4)} \wedge v(f_1(y))^{(5)} \wedge v(y)^{(6)}. \\
(3) & \leftarrow a(x_1, y_1, 0)^{(7)} \wedge e_1(x_1, f_1(y))^{(8)} \wedge e_0(y_1, y)^{(9)} \wedge \\
& b(f_1(y), v_1, 2)^{(10)} \wedge b(v_2, y, 2)^{(11)}. \quad \dagger_{10}, \dagger_{11} \\
(4) & \leftarrow b(f_0(y_1), y_1, 0)^{(12)} \wedge v(f_0(y_1))^{(13)} \wedge v(y_1)^{(14)}. \\
(5) & \leftarrow a(x_2, y_2, 1)^{(15)} \wedge e_0(x_2, f_0(y_1))^{(16)} \wedge e_\epsilon(y_2, y_1)^{(17)} \wedge b(f_0(y_1), v_1, 0)^{(18)} \wedge \\
& b(v_2, y_1, 0)^{(19)}. \quad \dagger_{18}, \dagger_{19} \\
(6) & \leftarrow b(y_1, y_1, 1)^{(20)} \wedge v(y_1)^{(21)}. \\
(7) & \leftarrow a(x, x, 0)^{(22)} \wedge e_1(x, f_1(x))^{(23)} \wedge e_1(x, f_1(x))^{(24)} \wedge b(y_1, v_1, 1)^{(25)}. \quad \dagger_{25} \\
(8) & \leftarrow a(x, x, 0)^{(22)} \wedge v(x)^{(26)}.
\end{aligned}$$

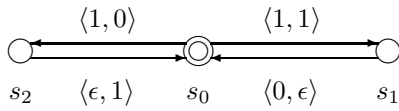
(a)

$$\begin{aligned}
(0) & \leftarrow b(x, x, 0)^{(0)}. \\
(1) & \leftarrow a(x, y, 2)^{(1)} \wedge e_\epsilon(x, z)^{(2)} \wedge [e_1(y, z)^{(3)}]. \\
(2) & \leftarrow b(x, y, 2)^{(4)} \wedge v(x)^{(5)} \wedge [e_1(y, x)^{(3)}]. \\
(3) & \leftarrow a(x_1, y_1, 0)^{(7)} \wedge e_1(x_1, x)^{(8)} \wedge e_0(y_1, y)^{(9)} \wedge b(x, v_1, 2)^{(10)} \wedge [e_1(y, x)^{(3)}]. \quad \dagger_{10} \\
(4) & \leftarrow a(x_1, y_1, 0)^{(7)} \wedge e_1(x_1, x)^{(8)} \wedge [e_0(y_1, y)^{(9)}] \wedge \cancel{[e_1(y, x)^{(3)}]}. \\
(5) & \leftarrow b(y, y_1, 0)^{(12)} \wedge v(y)^{(14)} \wedge [e_0(y_1, y)^{(9)}]. \\
(6) & \leftarrow a(x_2, y_2, 1)^{(15)} \wedge e_0(x_2, y)^{(16)} \wedge e_\epsilon(y_2, y_1)^{(17)} \wedge b(y, v_1, 0)^{(18)} \wedge \cancel{[e_0(y_1, y)^{(9)}]}. \quad \dagger_{18} \\
(7) & \leftarrow b(y_1, y_1, 1)^{(20)} \wedge v(y_1)^{(21)}. \\
(8) & \leftarrow a(x_3, y_3, 0)^{(22)} \wedge e_1(x_3, y_1)^{(23)} \wedge e_1(y_3, y_1)^{(24)} \wedge b(y_1, v_1, 1)^{(25)}. \quad \dagger_{25} \\
(9) & \leftarrow a(x_3, x_3, 0)^{(22)} \wedge v(x_3)^{(26)}.
\end{aligned}$$

(b)

Fig. 2. Resolution Proof for Example 4.9.

the state s_0 no sooner than all symbols of a domino have been accepted. For the instance of Example 4.9, we thus have an automaton with three states.



The encoding again allows to show the undecidability of our query rewriting problem (A PCP instance is satisfiable iff the maximally contained rewriting of $q \leftarrow b(x, x, 0)$. under Σ is nonempty.) as well as the undecidability of query containment under a set of cind's. (A PCP instance is satisfiable if and only if $\Sigma \models \{ \langle \rangle \mid \exists x : v(x) \wedge a(x, x, 0) \} \subseteq \{ \langle \rangle \mid \exists x : b(x, x, 0) \}$.)

Of course this correspondence between function-free and general resolution proofs does not hold for Horn clauses in general.

Example 4.10 The boolean query $q \leftarrow a_1(u, v) \wedge b_1(u, v)$. and the clauses

$$\begin{aligned}
a_1(f(x), y) & \leftarrow a_2(x, y). & a_2(x, g(y)) & \leftarrow a_3(x, y). \\
b_1(x, g(y)) & \leftarrow b_2(x, y). & b_2(f(x), y) & \leftarrow b_3(x, y).
\end{aligned}$$

taken together entail $q \leftarrow a_3(x, y) \wedge b_3(x, y)$. even though one cannot arrive at a function-free intermediate rewriting by either unfolding the left subgoal (resulting in $q \leftarrow a_2(x, y) \wedge b_1(f(x), y)$.) or the right subgoal (which would result in $q \leftarrow a_1(x, g(y)) \wedge b_2(x, y)$.) of our query first, neither by unfolding both at once (resulting in $q \leftarrow a_2(x, g(y)) \wedge b_2(f(x), y)$.) \square

5. Implementation

Our implementation is based on Algorithm 4.3, but makes use of several optimizations. Directly after parsing, Horn clauses whose head predicates are unreachable from the predicates of the query are filtered out. The same is done with clauses not in the set X computed by

```

X := ∅;
do X := X ∪
    {c ∈ C | Preds(Body(c)) ⊆
      (Sources ∪
       {Pred(Head(c')) | c' ∈ X})};
while X changed;

```

We have implemented the simple optimizations known from the Bucket Algorithm [20] and the Inverse Rules Algorithm [12] for answering queries using views which are used to reduce the branching factor in the search process. Beyond that, MiniCon descriptions are computed with an intelligent backtracking method that always chooses to cover subgoals first for which this can be done deterministically (i.e., the number of Horn clauses that are candidates for unfolding with a particular subgoal can be reduced to one), thereby reducing the amount of branching.

In the implementation of the deterministic component of our algorithm for generating MiniCon descriptions, we first check whether the corresponding pairs of terms of two atoms to match unify independently before doing full unification. This allows to detect most violations with very low overhead. Given an appropriate implementation, it is possible to check this property in logarithmic or even constant time.

Our unification algorithm allows to pre-specify variables that may in no case be unified with a function term (e.g., for head variables of queries or atoms already over source predicates). This allows to detect the impossibility to create a function-free rewriting as early as possible.

Every time an MCD m is unfolded with a query to produce an *intermediate* rewriting Q , we compute a query Q' (a partial rewriting) as

$$\begin{aligned} \text{Body}(Q') &:= \{\text{Body}_i(Q) \mid m_i \neq \epsilon\} \\ \text{Head}(Q') &:= \langle x_1, \dots, x_n \rangle \end{aligned}$$

s.t. $x_i \in \text{Vars}(\text{Head}(Q)) \cap \text{Vars}(\text{Body}(Q'))$ for $1 \leq i \leq n$. Q' is thus created from the new subgoals of the query that have been introduced using the MCD. If Q' contains non-source predicates, the following check is performed. We check if our rewriting algorithm produces a nonempty rewriting on Q' . This is carried out in depth-first fashion. If the set of cind's is cyclic, we use a maximum lookahead distance to assure that the search terminates. If Q' is not further rewritable, Q does not need to be further processed but can be dropped. Sub-

sequently, (intermediate) rewritings produced by unfolding queries with MiniCon descriptions are simplified using tableau minimization.

In Algorithm 4.3, the fact that MCDs are only applied one at a time causes a performance issue. It leads to redundant rewritings as e.g. the same MCDs may be applicable in different orders (as is true for the classical problem of answering queries using views, a special case) and thus a search space that may be larger than necessary. We use dependency graph-based optimizations to check if a denser packing of MCDs is possible. For the experiments with layered sets of cind's reported on in Section 6 (Figures 4 and 5), MCDs are packed exactly as densely as in the MiniCon algorithm of [22].

Distribution.

The implementation of our query rewriter consists of about 9000 lines of C++ code. Binaries for several platforms as well as examples and a Web demonstrator that allows to run limited-size problems online are available on the Web at [14].

6. Experiments

A number of experiments have been carried out to evaluate the scalability of our implementation. These were executed on a 600 MHz dual Pentium III machine running Linux. A benchmark generator was implemented that randomly generated example chain queries and sets of chain cind's⁶. Chain queries are conjunctive queries of the form

$$q(x_1, x_{n+1}) \leftarrow p_1(x_1, x_2) \wedge p_2(x_2, x_3) \wedge \dots \wedge p_{n-1}(x_{n-1}, x_n) \wedge p_n(x_n, x_{n+1}).$$

Thus, chain queries are constructed by connecting binary predicates via variables to form chains, as shown above. In our experiments, the distinguished (head) variables were the first and the last. The chain cind's had between 3 and 6 subgoals in both the subsuming and the subsumed queries.

In all experiments, the queries had 10 subgoals, and we averaged timings over 50 runs. Sets of cind's were always acyclic. This was ascertained by the use of predicate indexes such that the

⁶Experiments with various kinds of random queries and constraints were carried out, too. In this article, we only report on chain queries, but the experiments with random queries were similarly favorable.

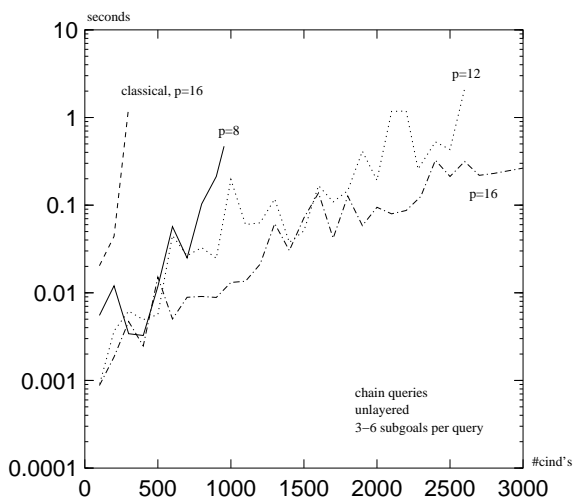


Fig. 3. Experiments with chain queries and nonlayered chain cind's.

predicates in a subsumer query of a cind only used indexes greater than or equal to a random number determined for each cind, and subsumed queries only used indexes smaller than that number. Times for parsing the input were excluded from the diagrams, and redundant rewritings were not eliminated⁷. Diagrams relate reasoning times on the (logarithmic-scale) vertical axis to the problem size expressed by the number of cind's on the horizontal axis.

We report on three experiments.

Figure 3 shows timings for non-layered sets of constraints. By the steep line on the left we report on an alternative query rewriting algorithm that we have implemented and which follows a traditional resolution strategy. This algorithm (evaluated using instances with 16 predicates) is compared to and clearly outperformed by our new algorithm (with three different numbers of predicates; 8, 12, and 16). Clearly, the more predicates are available, the sparser the constraints get. Thus, more predicates render the query rewriting process simpler.

In Figure 4, we report on the execution times of our new algorithm with cind's generated with an implicit layering⁸ of predicates (with 2 layers). This experiment is in principle very similar

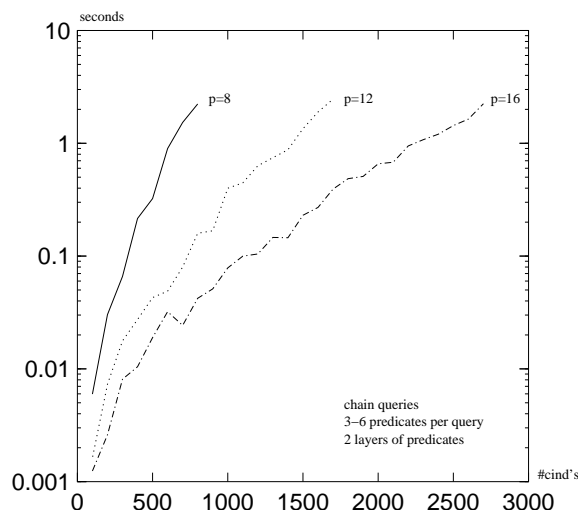


Fig. 4. Experiments with chain queries and two layers of chain cind's.

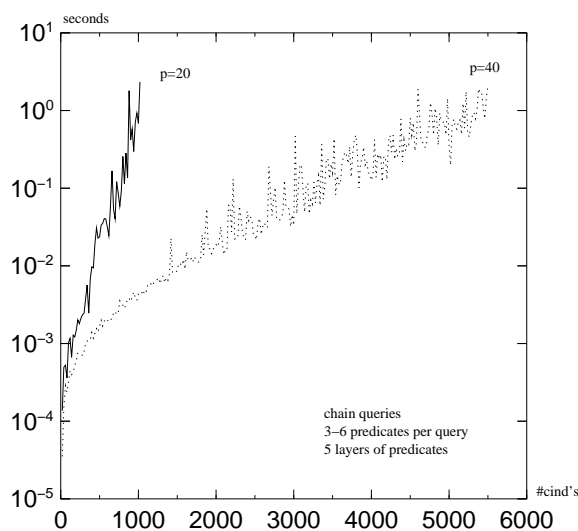


Fig. 5. Experiments with chain queries and five layers of chain cind's.

to local-as-view rewriting with $p/2$ global predicates and $p/2$ source predicates (where the subsumer queries of cind's correspond to logical views in the problem of answering queries using views), followed by view unfolding to account for the subsumed sides of cind's. We again report timings for three different total numbers of predicates, 8, 12, and 16.

In Figure 5, the new algorithm computes maximally contained rewritings for 20 and 40 predicates, which are grouped into stacks of five layers

⁷Note that our implementation optionally can make finite rewritings non-redundant and minimal. However, for our experiments, these options were not active.

⁸See Section 2 for our definition of layered sets of cind's.

of 4 and 8 predicates each, respectively. Of the five sets of predicates, one constitutes the sources and one the “integration schema” over which queries are asked, and four equally sized sets of cind’s bridge between these layers.

As can be seen by comparing the second and third diagrams with the first, the hardness of the layered problems is more homogeneous. Particularly in Figure 3 and Figure 4, one can also observe subexponential performance. Note that in the experiment of Figure 5, timings were taken in steps of 20 cind’s, while in the other experiments, this step length was 100.

7. Discussion and Conclusions

This article has addressed the query rewriting problem in data integration from a fresh perspective. We compute maximally contained rewritings with expressive symmetric constraints, which we call Conjunctive Inclusion Dependencies. We have proposed a new query rewriting algorithm based on techniques developed for the problem of answering queries using views (i.e., the MiniCon algorithm), which allows to apply time-tested (e.g., tableau minimization) techniques and algorithms from the database field to the query rewriting problem.

The main advantage of the new algorithm is that intermediate results are (function-free) queries and can be immediately made subject to query optimization techniques. As a consequence, further query rewriting may start from simpler queries, leading to an increase in performance and fewer redundant results that have to be found and later be eliminated. Thus, it is often possible to detect dead ends early. As a trade-off (as can be seen in Algorithm 4.3), an additional degree of nondeterminism is introduced compared to resolution-based algorithms that may temporarily introduce function terms.

In the context of data integration, there are usually a number of regularities in the way constraints are implemented and queries are posed. We expect to have a number of schemata, each one containing a number of predicates. Between the predicates of one schema, no constraints for data integration uses are defined. Moreover, we expect inter-schema constraints to be of the form $Q_1 \subseteq Q_2$ where most (or all) predicates in Q_1 belong to one and the

same schema, while the predicates of Q_2 belong to another one. Queries issued against the system are usually formulated in terms of a single schema, and such a layering often propagates along intermediate rewritings. Given these assumptions, we suspect our approach – when optimization techniques from the database area are applied to intermediate results – to have a performance advantage over classical resolution-based algorithms, which do not exploit such techniques.

Our experiments show that our approach scales to very large and complex data integration settings with many schemata.

References

- [1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [2] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. “Query Caching and Optimization in Distributed Mediator Systems”. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD’96)*, pages 137–146, Montreal, Canada, June 1996.
- [3] Y. Arens and C. A. Knoblock. “Planning and Reformulating Queries for Semantically-Modeled Multi-database Systems”. In *Proceedings of the First International Conference on Information and Knowledge Management (CIKM’92)*, Baltimore, MD USA, 1992.
- [4] D. Calvanese, G. De Giacomo, and M. Lenzerini. “On the Decidability of Query Containment under Constraints”. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS) 1998*, pages 149–158, 1998.
- [5] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. “Information Integration: Conceptual Modeling and Reasoning Support”. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS’98)*, pages 280–291, New York City, New York, USA, 1998.
- [6] M. Carey, L. Haas, P. Schwarz, M. Arya, W. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. Williams, and E. Wimmers. “Towards Heterogeneous Multimedia Information Systems: The Garlic Approach”. In *Proceedings of the 5th International Workshop on Research Issues in Data Engineering: Distributed Object Management (RIDE-DOM’95)*, 1995.
- [7] E. Dantsin and A. Voronkov. “Complexity of Query Answering in Logic Databases with Complex Values”. In *LFCS’97, LNCS 1234*, pages 56–66, 1997.
- [8] N. Dershowitz and J.-P. Jouannaud. “Rewrite Systems”. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 2, chapter 6, pages 243–320. Elsevier Science Publishers B.V., 1990.

- [9] O. M. Duschka and M. R. Genesereth. “Answering Recursive Queries using Views”. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 11–15, 1997, Tucson, AZ USA, Tucson, AZ USA, 1997*.
- [10] O. M. Duschka, M. R. Genesereth, and A. Y. Levy. “Recursive Query Plans for Data Integration”. *Journal of Logic Programming*, 43(1):49–73, 2000.
- [11] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. “The TSIMMIS Approach to Mediation: Data Models and Languages”. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.
- [12] M. R. Genesereth, A. M. Keller, and O. M. Duschka. “Infomaster: An Information Integration System”. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (SIGMOD’97)*, pages 539–542, 1997.
- [13] J. Gryz. “Query Rewriting Using Views in the Presence of Functional and Inclusion Dependencies”. *Information Systems*, 24(7):597–612, 1999.
- [14] C. Koch. “Cindrew Home Page”, 2001. <http://cern.ch/chkoch/cindrew/>.
- [15] C. Koch. “Data Integration against Multiple Evolving Autonomous Schemata”. PhD thesis, TU Wien, Vienna, Austria, 2001.
- [16] C. Koch. “Query Rewriting with Symmetric Constraints”. In *Proc. 2nd International Symposium on Foundations of Information and Knowledge Systems (FoIKS)*, pages 130–147, Salzau Castle, Germany, Feb. 2002. Springer LNCS 2284.
- [17] C. T. Kwok and D. S. Weld. “Planning to Gather Information”. In *Proceedings of the 13th National Conference on Artificial Intelligence, AAAI’96*, pages 32–39, Portland, OR USA, Aug. 1996.
- [18] M. Lenzerini. “Data Integration: A Theoretical Perspective”. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS) 2002*, 233–246.
- [19] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. “Answering Queries Using Views”. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS) 1995*, pages 95–104, San Jose, CA USA, 1995.
- [20] A. Y. Levy, A. Rajaraman, and J. J. Ordille. “Querying Heterogeneous Information Sources Using Source Descriptions”. In *Proceedings of the 1996 International Conference on Very Large Data Bases (VLDB’96)*, pages 251–262, 1996.
- [21] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [22] R. Pottinger and A. Y. Levy. “A Scalable Algorithm for Answering Queries Using Views”. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB’2000)*, pages 484–495, 2000.
- [23] X. Qian. “Query Folding”. In *Proceedings of the 12th IEEE International Conference on Data Engineering (ICDE’96)*, pages 48–55, New Orleans, LA USA, 1996.
- [24] M. F. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.
- [25] S. Vorobyov and A. Voronkov. “Complexity of Non-recursive Logic Programs with Complex Values”. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS) 1998*, 1998.