Tractable Database Design through Bounded Treewidth

Georg Gottlob, Reinhard Pichler, and Fang Wei
Database and Artificial Intelligence Group
Technische Universität Wien
A1040 Vienna, Austria
[gottlob | pichler | wei]@dbai.tuwien.ac.at

ABSTRACT

Given that most elementary problems in database design are NP-hard, the currently used database design algorithms produce suboptimal results. For example, the current 3NF decomposition algorithms may continue further decomposing a relation even though it is already in 3NF. In this paper we study database design problems whose sets of functional dependencies have bounded treewidth. For such sets, which frequently occur in practice, we develop polynomial-time and highly parallelizable algorithms for a number of central database design problems such as:

- primality of an attribute
- 3NF-test for a relational schema or subschema
- BCNF-test for a subschema.

For establishing these results, we propose a new characterization for keys and for the primality of a single attribute.

In order to define the treewidth of a relational schema, we shall associate a hypergraph with it. Note that there are two main possibilities of defining the treewidth of a hypergraph H: One is via the primal graph of H and one is via the incidence graph of H. Our algorithms apply to the case where the primal graph is considered. However, we also show that the tractability results still hold when the incidence graph is considered instead.

Categories and Subject Descriptors: H.2.1 [**Logical Design**]: Data models, Normal forms, Schema and subschema; F.1.2 [**Modes of Computation**]: Alternation and nondeterminism, Parallelism and concurrency; F.2.2 [**Nonnumerical Algorithms and Problems**]: Complexity of proof procedures, Computations on discrete structures

General Terms: Algorithms, Design, Theory

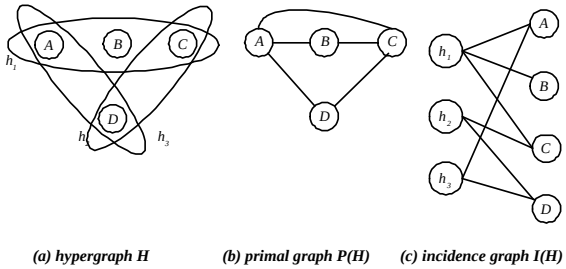
Keywords: Normal forms, Database design, Tree decomposition, Bounded treewidth, Fixed-parameter tractability

1. INTRODUCTION

One of the fundamental problems in relational database design is to test whether a schema satisfies the desired normal form. Unfortunately, most problems arising in this area are intractable. For instance, among the following six most important decision problems:

- PRIMALITY (for a schema or for a subschema)
- 3NFTEST (for a schema or for a subschema)
- BCNFTEST (for a schema or for a subschema)

only BCNFTEST for a schema is tractable [18]. Despite the intractability of the remaining problems listed above, there exist efficient algorithms for decomposing a relational schema into subschemas satisfying 3NF or BCNF (see e.g., [18, 5, 19]. However, without the ability of normal form checking, they are doomed to produce suboptimal results even in very simple situations. Most unsatisfactorily, these decomposition algorithms may continue further decomposing a schema or subschema even though the desired normal form has already been achieved. The following example displays a simple relational schema, where the well-known 3NF decomposition algorithm *synthesis* from [5] keeps decomposing the schema even though it is already in 3NF.



(a) hypergraph H *(b) primal graph P(H)* *(c) incidence graph I(H)*

Figure 1: (Hyper-)graphs from Example 1.2.

EXAMPLE 1.1. *Consider the schema $ABCD$ with the FDs $AB \to C, C \to D, D \to A$. This schema is already in 3NF since all its attributes are prime. However by using the synthesis algorithm, we obtain a decomposition ABC, CD, DA.*

The concept of treewidth and related notions have been successfully applied to many areas of Computer Science. Recently, several intractable problems in the Database field and in AI (such as e.g., conjunctive query equivalence and CSP problems) have been shown to become solvable in polynomial time or even highly parallelizable if the underlying graph or hypergraph structure has bounded treewidth or hypertree width (see [16]). Hence, the question naturally arises as to whether one can also identify such tractable fragments for the aforementioned decision problems in relational database design. In this work, we define the treewidth of a relational schema (R, F) (where F denotes the set of

PODS'06, June 26–28, 2006, Chicago, Illinois, USA.

functional dependencies holding in R) by considering the following hypergraph H: the attributes of R are the vertices of H and every hyperedge of H corresponds to the set of attributes occurring in a functional dependency $f \in F$. In order to define the treewidth of H, one can either consider the *primal graph* $P(H)$ or the *incidence graph* $I(H)$. In this paper, we shall mainly deal with the primal graph.

EXAMPLE 1.2. *Consider the relational schema from Example 1.1. The corresponding hypergraph H consists of the vertices $\{A, B, C, D\}$ and the hyperedges ABC, CD, AD. The hypergraph H plus the primal graph $P(H)$ and the incidence graph $I(H)$ are shown in Figure 1. Note that the treewidth both of $P(H)$ and of $I(H)$ is 2.*

One possibility to prove the fixed-parameter tractability of the above mentioned decision problems is to show that these problems can be expressed in terms of MSO formulae over the primal graph or over the incidence graph, respectively. The fixed-parameter tractability is thus an immediate consequence of Courcelle's Theorem [8] (see Section 8). A concrete algorithm can then be obtained by constructing a finite tree automaton (FTA) corresponding to the MSO formula and by checking whether a tree obtained from the tree decomposition is accepted by the FTA. However, algorithms resulting from such an MSO-to-FTA transformation often have two serious shortcomings: First, the intuition of the algorithm is usually lost when looking at the FTA. Second (and even more importantly), the FTA resulting from standard translation methods (see e.g., [13]) often suffers from a state explosion and tends to be unnecessarily complicated. For these reasons, it is always preferable to have a dedicated algorithm rather than just an MSO-encoding.

The main contributions of this paper are the following:

- For all of the above mentioned database design problems (i.e., PRIMALITY, 3NFTEST, and BCNFTEST – both for a schema and for a subschema), we manage to identify tractable fragments via bounded treewidth. In case of bounded treewidth of the primal graph, we develop new, dedicated algorithms, from which the fixed-parameter tractability follows immediately. Actually, even if no tree decomposition is given, all of these problems are not only tractable but also highly parallelizable since they are in the class LOGCFL for either notion of bounded treewidth.

- For the case of bounded treewidth of the incidence graph, we establish the fixed-parameter tractability by providing an MSO-encoding of these problems.

- The canonical definition of PRIMALITY (i.e, the attribute is contained in some key) turns out to be impractical for our new algorithms. Hence, we introduce the concept of "co-antikeys" which allows for a new characterization of PRIMALITY. Moreover we present a new method which allows one to divide the problem of computing the keys of a relational schema (or, by the same token, to divide the problem of deciding primality in a relational schema) into smaller subproblems. Of course, by the intrinsic exponentiality of the set of keys in a relational schema, we cannot expect to achieve tractability. Nevertheless our new method constitutes an efficient to implement heuristics (via well-studied graph problems), which may help to greatly simplify the problems of computing the keys and of testing primality in many situations.

The rest of the paper is organized as follows: In Section 2 we recall some basic terminology and results. A characterization of primality and a simplification of computing keys are presented in Sections 3 and 4. In the Sections 5 through 7 we present new algorithms for the above mentioned six decision problems in database design. In Section 8, we prove the corresponding fixed-parameter tractability results also for the case of bounded treewidth of the incidence graph via appropriate MSO-encodings. The application of these results to normal form decompositions is dealt with in Section 9. We conclude with Section 10.

2. PRELIMINARIES

2.1 Database Design

A relational schema is denoted as (R, F) where R is the set of attributes, and F the set of functional dependencies (FDs, for short) over R. If X and Y are sets of attributes and A is an attribute, then we may write XY, XA, $X - A$ to abbreviate the set notation $X \cup Y$, $X \cup \{A\}$, $X \setminus \{A\}$.

Unless explicitly stated otherwise, we only consider FDs in *canonical form* here, i.e., FDs where there is only a single attribute on the right-hand side. The set of all FDs over the attributes in R that can be derived from F via *Armstrong's Axioms* (see [1]) is denoted as F^+. We write $F \models X \rightarrow A$ in order to denote that an FD $X \rightarrow A$ is in F^+.

Given a relational schema (R, F) and a subset $X \subseteq R$, we write $clos_F(X)$ for the *closure* of X with respect to F, i.e., $A \in clos_F(X)$, iff $F \models X \rightarrow A$. If $F \models X \rightarrow A$, then we also say that "X determines (or decides) A".

If X determines all attributes $A \in R$, then X is called a *superkey*. If X is minimal with this property, then X is a *key*. The set of all keys in (R, F) is denoted as $K(R, F)$. An attribute A is called *prime* in (R, F), if it is contained in at least one key in $K(R, F)$.

In this paper, we often have to refer to the attributes occurring in an FD f. Let f be of the form $f = X \rightarrow A$. Then we write $lhs(f)$ and $rhs(f)$ to refer to X and A, respectively.

A "derivation sequence" of A from X in F is a sequence of the form $X \rightarrow X \cup \{A_1\} \rightarrow X \cup \{A_1, A_2\} \rightarrow \ldots \rightarrow X \cup \{A_1, \ldots, A_n\}$, s.t. $A_n = A$ and for every $i \in \{1, \ldots, n\}$, there exists an FD $f_i \in F$ with $lhs(f) \subseteq X \cup \{A_1, \ldots, A_{i-1}\}$ and $rhs(f) = A_i$. Of course, such a derivation sequence exists, iff $A \in clos_F(X)$.

DEFINITION 2.1. Let (R, F) be a relational schema and $X \subseteq R$. A schema projection of F over X is defined as

$$F[X] = \{Y \rightarrow Z \mid F \models Y \rightarrow Z \text{ and } YZ \subseteq X\}$$

Let $R' \subseteq R$. Then $(R', F[R'])$ is referred to as a *subschema* of (R, F). Note that the size of $F[R']$ can be exponentially bigger than (R, F) (see [18]). Hence, for the complexity of an algorithm that deals with subschemas, it is important to distinguish whether $F[R']$ is explicitly given or whether it is only implicit in R'. In all our algorithms here, we assume that it is implicit in R'.

2.2 Tree Decompositions and Treewidth

A *hypergraph* is a pair $\mathcal{H} = \langle V, H \rangle$ consisting of a set V of vertices and a set H of hyperedges. A hyperedge $h \in H$ is a subset of V. The *primal graph* $P(\mathcal{H})$ (also called the Gaifman graph) has the same set of vertices as H. Moreover, two vertices v_i, v_j are connected in $P(\mathcal{H})$ if they jointly occur in some hyperedge $h \in H$. On the other hand, the *incidence graph* $I(\mathcal{H})$ is a bipartite graph with vertices $V \cup H$ (i.e., the

vertices of $I(\mathcal{H})$ are the vertices v_i of H plus the hyperedges h_j of H). Two vertices v_i and h_j are connected in $I(\mathcal{H})$ if in $\mathcal{H}$, the vertex v_i occurs in the hyperedge h_j.

A measure for the "tree-likeness" of a graph $\mathcal{G} = \langle V, E\rangle$ is the treewidth of $\mathcal{G}$, which we shall define below. A *tree decomposition* $\mathcal{T}$ of $\mathcal{G}$ is a pair $\langle T, \lambda\rangle$, where T is a tree and λ is a labeling function with $\lambda(N) \subseteq$V for every node $N \in T$, s.t. the following conditions hold:

1. $\forall v \in V$, there exists a node N in T, s.t. $v \in \lambda(N)$.
2. $\forall e \in E$, there exists a node N in T, s.t. $e \subseteq \lambda(N)$.
3. "connectedness condition": $\forall v \in V$, the set of nodes $\{N \mid v \in \lambda(N)\}$ induces a connected subtree of T.

The sets $\lambda(N)$ for the nodes N in $\mathcal{T}$ are referred to as *bags*. The *width* of a tree decomposition $\langle T, \lambda\rangle$ is $\max(\{|\lambda(N)|-1 : N \text{ node in } T\})$. Let $\mathcal{G} = \langle V, E\rangle$ be a graph. The *tree-width* $tw(\mathcal{G})$ is the minimum width over all its tree decompositions. The treewidth $tw(\mathcal{H})$ of a hypergraph $\mathcal{H}$ can be either defined as the treewidth of the primal graph $P(\mathcal{H})$ or of the incidence graph $I(\mathcal{H})$. We thus set $tw_P(\mathcal{H}) = tw(P(\mathcal{H}))$ and $tw_I(\mathcal{H}) = tw(I(\mathcal{H}))$, respectively.

In order to define the treewidth of a relational schema (R, F) or of a finite structure $\mathfrak{A}$, we just have to indicate which hypergraph we are considering.

DEFINITION 2.2. For a relational schema (R, F), we define the *hypergraph* of (R, F) as $\mathcal{H} = \langle V, H\rangle$ with $V = R$ and $H = \{\{A_1, \ldots, A_n, B\} \mid (A_1 \ldots A_n \rightarrow B) \in F\}$.

The primal graph and the incidence graph of (R, F) are simply defined as the corresponding graph of $\mathcal{H}$. Likewise, the *treewidth* of (R, F) is defined as $tw_P(\mathcal{H})$ or as $tw_I(\mathcal{H})$, respectively. Actually, in this paper, we only consider the *treewidth via the primal graph* – the only exception being Section 8, which is devoted to the incidence graph. Hence, outside Section 8, we shall simply speak about the treewidth (or a tree decomposition, resp.) of a relational schema (R, F) in order to refer to the treewidth (or a tree decomposition, resp.) of the *primal graph* of $\mathcal{H}$.

Let $\mathcal{T}$ be a tree decomposition of the primal graph of $\mathcal{H}$, N a node in $\mathcal{T}$ and $f = A_1 \ldots A_n \rightarrow B$ an FD in F. We say that f is *covered* by a node N, if $\{A_1, \ldots, A_n, B\} \subseteq \lambda(N)$.

DEFINITION 2.3. Let $\mathfrak{A}$ be a finite structure with universe A. The *hypergraph* corresponding to $\mathfrak{A}$ is defined as $\mathcal{H} = \langle V, H\rangle$ with $V = A$ and $H = \{\{a_1, \ldots, a_n\} \mid \mathfrak{A}$ contains a relational ground atom $P(a_1 \ldots a_n)$ for some predicate symbol $P\}$.

The primal graph and the incidence graph of $\mathfrak{A}$ as well as the notions of *tree decomposition* and *treewidth* of $\mathfrak{A}$ are then defined in the obvious way via the hypergraph $\mathcal{H}$.

2.3 Complexity

The class LOGCFL consists of all those decision problems that are log-space reducible to a context-free language. The relationship between LOGCFL and other well-known complexity classes is summarized as follows:

$\mathbf{L} \subseteq \mathbf{NL} \subseteq$ LOGCFL $\subseteq \mathbf{AC}^1 \subseteq \mathbf{NC}^2 \subseteq \mathbf{P}$

Since LOGCFL $\subseteq \mathbf{AC}^1 \subseteq \mathbf{NC}^2$, the problems in LOGCFL are highly parallelizable.

3. CHARACTERIZATION OF PRIMALITY

We shall provide a new characterization of the primality of an attribute via the concept of antikeys (also known as maximal nonkeys) introduced in [9].

DEFINITION 3.1. [9] Given a relational schema (R, F), $X \subset R$ is an antikey, if X is not a key, and for any $A \in R \setminus X$, $X \cup \{A\}$ is a key.

We write antikeys(R, F) to denote the set of all the antikeys of a relational schema (R, F), and co-antikeys(R, F) for the complements of the antikeys, i.e.

$$\text{co-antikeys}(R, F) = \{R \setminus W \mid W \in \text{antikeys}(R, F)\}$$

K(R, F) can be considered as a hypergraph (whose hyperedges are the keys of (R, F). Likewise, co-antikeys(R, F) can be considered as a hypergraph. The following relationship between these two hypergraphs was shown in [9]:

$$\text{K}(R, F) = \text{Tr}(\text{co-antikeys}(R, F))$$

where Tr(H) denotes the set of minimal transversals of H.

EXAMPLE 3.2. *Consider the schema ABCD with the FDs $AB \rightarrow C, C \rightarrow D, D \rightarrow A$ from Example 1.1. There are two antikeys B and ACD, and the corresponding co-antikeys are ACD and B. The keys are AB, BC and BD.*

As was mentioned above, all the keys of a given relational schema (R, F) can be obtained by generating the minimal transversals of co-antikeys(R, F). Hypergraph transversal and related problems have been intensively studied in recent years [11, 12, 14]. However, it is still an open problem whether an output-polynomial algorithm exists. Fortunately, in order to decide whether a given attribute A is prime, we only need to check whether A is an element of some co-antikey.

Recall that a hypergraph is simple, if it has no pair of edges h_i, h_j such that h_i is properly contained in h_j. The following property holds in simple hypergraphs.

THEOREM 3.3. *Let $\mathcal{H}$ be a simple hypergraph with hyperedges $H = \{h_1, \ldots, h_n\}$. Then each node in $h_1, \ldots, h_n$ exists in at least one minimal transversal of $\mathcal{H}$.*

Proof. For simple hypergraphs, the following property of transversals holds [4]: $\text{Tr}(\text{Tr}(\mathcal{H})) = \mathcal{H}$. Assume that there is one vertex $x \in \mathcal{H}$, such that $x \notin \text{Tr}(\mathcal{H})$, then $x \notin \text{Tr}(\text{Tr}(\mathcal{H}))$, which means that $x \notin \mathcal{H}$, which is a contradiction. □

LEMMA 3.4. *The hypergraph of the co-antikeys of any relational schema is simple.*

Proof. The set of antikeys is a Sperner system (or an antichain)[9]. It follows from the well-known property of the Sperner system, stating that no element in a Sperner system is contained in another. □

We thus get a new characterization of primality:

THEOREM 3.5. *Let (R, F) be a relational schema and $A \in R$. A is a prime attribute, if and only if there is a co-antikey R' of (R, F) which contains A.*

Proof. This follows from Theorem 3.3 and Lemma 3.4. □

We now also provide a new characterization of the co-antikeys. For this purpose, we introduce the new concept of a *shadow subschema*.

DEFINITION 3.6. Let (R, F) be a relational schema and $X \subseteq R$. We define the *shadow schema projection* of F over X as follows:

$$F_X = \{U \rightarrow A \mid (W \rightarrow A \in F) \wedge (U = W \cap X)\}$$

The schema (X, F_X) is called the corresponding *shadow subschema* of (R, F).

In order to simplify the presentation, we sometimes write (R', F') to denote the *shadow subschema* $(R', F_{R'})$. We want to emphasize that the *shadow schema projection* F_X introduced here is essentially different from the schema projection $F[X]$ defined in Definition 2.1 in that from each FD rule, the *shadow schema projection* F_X simply projects out the attributes which do not exist in X. The size of F_X is thus clearly bounded by the size of F. This is in great contrast to $F[X]$, which may be exponentially bigger than F (see [18]). Another important difference from $F[X]$ is that F_X may in fact contain FDs which are not contained in F.

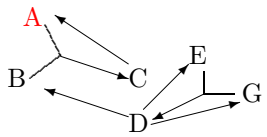


Figure 2: FDs of Example 3.7

EXAMPLE 3.7. Consider the relational schema with attributes $ABCDEG$ and FDs $F = \{AB \to C, C \to A, D \to B, EG \to D, D \to E, D \to G\}$ (see Figure 2).

Consider the subset $X \subset R$ with $X = DEG$. Then $F_X = \{EG \to D, D \to E, D \to G\}$. In this case – accidentally – F_X is a cover of $F[X]$.

Now consider $Y = DE$. Then $F_Y = \{E \to D, D \to E\}$. Note that the FD $E \to D$ is *not* contained in F.

The co-antikeys can be characterized as follows:

THEOREM 3.8. *Let (R, F) be a relational schema and let (R', F') be a shadow subschema where $F' = F_{R'}$. Then R' is a co-antikey if and only if the following conditions hold:*

1. *For each attribute $A \in R'$, $clos_{F'}(A) = R'$ and*
2. *$(\forall A \in R')\ F \not\models (R \setminus R') \to A$.*

Proof. The crucial observation is that in some situations, there is a one-to-one correspondence between derivation sequences in a schema (R, F) and in a shadow subschema $(R', F_{R'})$, The following lemma gives a sufficient criterion.

LEMMA 3.9. *Let (R, F) be a relational schema and let (R', F') be a shadow subschema where $F' = F_{R'}$, s.t. none of the attributes in R' is decided by $(R \setminus R')$, i.e., $(\forall A \in R')$ $F \not\models (R \setminus R') \to A$. Moreover, let $X \subseteq R'$ and $A \in R'$. Then $F' \models X \to A$, iff $F \models ((R \setminus R') \cup X) \to A$.* ◇

The lemma is shown by induction on the length k of derivation sequences. With this lemma at our disposal, the proof of the theorem is straightforward. □

4. SCC SIMPLIFICATION

In order to test whether some attribute A is prime in (R, F) it is sometimes sufficient to test whether A is prime in some shadow subschema of (R, F). This section provides a sufficient criterion for such a simplification. We call it SCC simplification (SCC stands for Strongly Connected Component). For this purpose we first establish the relationship between relational schemas and digraphs.

DEFINITION 4.1. Let (R, F) be a relational schema. The *digraph corresponding to* (R, F) is defined as $\mathcal{D}(R, F) = (R, E)$, where $(B, A) \in E$, iff there is an FD $X \to A \in F$, such that $B \in X$.

DEFINITION 4.2. Let $\mathcal{D}(R, F)$ be the digraph of the relational schema (R, F). The *strongly connected components* of (R, F) are the strongly connected components of $\mathcal{D}(R, F)$.

DEFINITION 4.3. A strongly connected source component of any digraph $G = (V, E)$ is a strongly connected component without any incoming edge.

Obviously, every digraph has at least one strongly connected source component.

The goal of our simplification algorithm will be to reduce the primality test of an attribute in some schema (R, F) to the primality test in some "smaller" (in a sense to be made precise below) schema. To this end, we shall provide a criterion for splitting R into disjoint subsets $R_1, \ldots, R_l$ s.t. the primality only has to be tested in some schema with attributes in one such subset R_i rather than in R. The kind of subsets $R_i \subset R$ that we are interested in are the *key components* defined below:

DEFINITION 4.4. Let (R, F) be a relational schema and let (R', F') be a shadow subschema of (R, F) where $R' \subseteq R$. (R', F') is a *key component* if for every FD $(X \to A) \in F$ the following holds: if $A \in R'$, then $X \subseteq R'$.

LEMMA 4.5. *Let (R, F) be a relational schema and let (R', F') be a key component of (R, F). Let further (R'', F'') be a shadow subschema of (R, F) with $R'' = R \setminus clos_F(R')$ and $F'' = F_{R''}$. Then the following relationship holds:*

$$K(R, F) = K(R', F') \oplus K(R'', F'')$$

where $\oplus$ is defined as follows:

$$X \oplus Y = \{xy \,|\, x \in X, y \in Y\}$$

Proof. The following property of key components has to be shown first: *Let (R, F) be a relational schema and (R', F') be a key component of (R, F). Assume $A \in R'$ and $X \subseteq R'$. Then the following holds: $(F' \models X \to A) \Rightarrow (F \models X \to A)$.* The proof of Lemma 4.5 is then easy. □

Lemma 4.5 gives us a hint how to construct an algorithm (see Figure 3) that decomposes (R, F) into key components.

THEOREM 4.6. *Let $(R_1, F_1), \ldots, (R_n, F_n)$ be all the output key components in the Algorithm of Figure 3. Then the following equality holds:*

$$K(R, F) = K(R_1, F_1) \oplus K(R_2, F_2) \oplus \ldots \oplus K(R_n, F_n)$$

Proof. In the first run of the loop, we obtain from Lemma 4.5 that $K(R, F) = K(R_1, F_1) \oplus K(R'_1, F'_1)$, where (R'_1, F'_1) is the shadow subschema of (R, F) and (R'_1, F'_1) is the input schema of the second run of the loop. The program clearly terminates. Successive application of Lemma 4.5 yields the following sequence of equalities:

$$\begin{aligned} &K(R, F) = K(R_1, F_1) \oplus K(R'_1, F'_1) \\ &K(R'_1, F'_1) = K(R_2, F_2) \oplus K(R'_2, F'_2) \\ &\ldots \\ &K(R'_l, F'_l) = K(R_{l+1}, F_{l+1}) \oplus \emptyset \end{aligned}$$

The claim of the theorem follows immediately. □

With the key components output by the algorithm, the primality test of any attribute can be restricted to a single key component:

THEOREM 4.7. *Let (R, F) be a relation schema and an attribute $A \in R$. A is prime in (R, F), iff A is prime in (R', F'), where (R', F') is some output component in the algorithm Key-Component-Generation.* □

EXAMPLE 4.8. Recall the relational schema (R, F) from Example 3.7 with $R = ABCDEG$ and $F = \{AB \to C, C \to A, D \to B, EG \to D, D \to E, D \to G\}$. We shall check whether A is prime via the Key-Component-Generation algorithm:

Running the Key-Component-Generation algorithm over the schema, we obtain the first key component with attributes $R_1 = DEG$. Since $clos_F(R_1) = BDEG$, we obtain $R'_1 = R \setminus BDEG = AC$. Note that the shadow subschema $(R'_1, F_{R'_1})$ contains the FDs $\{A \to C, C \to A\}$.

In the second iteration of the loop we have $R = AC$ and $F = \{A \to C, C \to A\}$. It is clear that the attributes AC constitute the second key component and the program halts.

Now we only need to check whether A is prime in the schema $(AC, \{A \to C, C \to A\})$, which indeed is the case.

There is a well-known linear time algorithm for finding all the *strongly connected components* of a digraph [22]. Likewise, the *closure generation algorithm* takes linear time [18]. Hence, the algorithm Key-Component-Generation can be executed efficiently. It is therefore clearly worth trying to apply the SCC simplification in order to reduce the primality problem to a strictly smaller subproblem. By the NP-completeness of primality, the worst-case complexity of testing primality is not affected by the SCC simplification. Nevertheless, it is an efficient heuristics which may help to reduce the problem size in many situations.

Algorithm Key-Components-Generation
Input relational schema (R, F)
Output $\mathcal{R} = \{(R_1, F_1), \ldots, (R_n, F_n)\}$ s.t. $(\forall i)\, R_i \subseteq R$
begin
  $\mathcal{R} := \emptyset$; $i := 0$
  Repeat
    Decompose $\mathcal{D}(R, F)$ into its strongly connected components;
    $i := i + 1$;
    Choose an arbitrary strongly connected source component (R_i, F_i) of $\mathcal{D}(R, F)$;
    $\mathcal{R} := \mathcal{R} \cup (R_i, F_i)$
    $R'_i := R \setminus clos_F(R_i)$;
    $(R'_i, F'_i) :=$ shadow subschema by R'_i;
    $(R, F) := (R'_i, F'_i)$;
  Until $R = \emptyset$;
end.

Figure 3: Key Components Generation Algorithm

5. BOUNDED TREEWIDTH & PRIMALITY

It has already been mentioned that testing whether some attribute A is prime in a relational schema (R, F) is NP-complete (see [17]). In this section we show that if the set of FDs has bounded tree-width,[1] the primality test becomes tractable. In fact, we even show that this problem falls into the highly parallelizable complexity class LOGCFL.

We start with an informal explanation of the Primality-Test algorithm in Figure 4. At the heart of this algorithm is the procedure prime-attribute, which is used to realize a top-down traversal of the tree decomposition $\mathcal{T}$, where the root can be any node N with $A \in \lambda(N)$. The goal of the algorithm is to successively guess all elements of a co-antikey $\mathcal{X}$ with $A \in \mathcal{X}$. At each node N_i in $\mathcal{T}$, we thus guess X_{N_i} in step 1 with the intended meaning $X_{N_i} = \lambda(N_i) \cap \mathcal{X}$. Of course, the guess at N_i has to be consistent with the guess at the parent N of N_i. This is ensured by condition 1.b. Note

[1]Recall that outside Section 8 we only consider the treewidth via the *primal graph*.

Algorithm Primality-Test
Input Relational schema (R, F), tree decomposition $\mathcal{T}$ of the primal graph of (R, F) with width k, attribute $A \in R$.
Output "Accept", if A is prime in (R, F).

Procedure check-primality (X_M: SetOfAttributes, Cov_M,P_M: OrderedSetOfFDs, M: Node in $\mathcal{T}$)
begin
  Check 1: $\forall f \in F$: If f is covered by $\lambda(M)$ and $rhs(f) \in X_M$ then $lhs(f) \cap X_M \neq \emptyset$,
  Check 2: $\forall g \in P_M$ with $g = U \to B$ and $B \notin U$:
    (i) Either $g = f_{X_M}$ for some $f \in F$ covered by $\lambda(M)$
    (ii) or there exists an FD $h = V \to B$ in Cov_M with $V \subset U$ and $h < g$
    (iii) or there exist FDs $h_1 = V \to C$ and $h_2 = WC \to B$ in Cov_M with $V \cup W \subseteq U$ and $h_1, h_2 < g$.
  if both Checks succeed **then Accept**
  else HALT and **Reject**;
end;

Procedure prime-attribute (X_N: SetOfAttributes, Cov_N, P_N: OrderedSetOfFDs, N_i: Node in $\mathcal{T}$)
begin
1) **Guess** $X_{N_i} \subseteq \lambda(N_i)$ s.t.:
  1.a) If $\lambda(N_i) \cap X_N = \emptyset$ then $X_{N_i} = \emptyset$,
  1.b) $\forall B \in \lambda(N) \cap \lambda(N_i)$: $B \in X_N \Leftrightarrow B \in X_{N_i}$
2) **Guess** Cov_{N_i} ordered set of FDs over X_{N_i} s.t.:
  2.a) $\forall B, C \in X_{N_i}$: $(B \to C) \in Cov_{N_i}$,
  2.b) $\forall$ FDs g over $\lambda(N) \cap \lambda(N_i)$: $g \in Cov_N \Leftrightarrow g \in Cov_{N_i}$
  2.c) The orderings on Cov_N and Cov_{N_i} are consistent.
3) $P_{N_i} := P_N \cup (Cov_{N_i} - Cov_N)$;
  3.a) **if not** $P_{N_i} \subseteq Cov_{N_i}$ **then HALT** and **Reject**;
4) **Guess** assignment $\alpha_{N_i} : P_{N_i} \to \{0, \ldots, n\}$, where $n =$ number of child nodes of N_i.
5) check-primality (X_{N_i}, Cov_{N_i}, $\alpha_{N_i}^{-1}(0)$, N_i);
6) **if** the check 5) fails then **Reject**;
7) **if for each** $\ell \in \{1, \ldots, n\}$
    prime-attribute (X_{N_i}, Cov_{N_i}, $\alpha_{N_i}^{-1}(\ell)$, M_ℓ) = Accept,
    where M_ℓ is the ℓ-th child of N_i in $\mathcal{T}$
  then Accept
  else Reject;
end;

begin (* MAIN *)
  r0) (* initializations *)
    find some node N in $\mathcal{T}$ with $A \in \lambda(N)$;
    consider $\mathcal{T}$ as rooted at N;
  r1) **Guess** $X_N \subseteq \lambda(N)$ with $A \in X_N$;
  r2) **Guess** Cov_N ordered set of FDs over X_N s.t.:
    r2.a) $\forall B, C \in X_N$: $(B \to C) \in Cov_N$.
  r3) $P_N := Cov_N$;
  r4) **Guess** assignment $\alpha_N : P_N \to \{0, \ldots, n\}$, where $n =$ number of child nodes of N.
  r5) check-primality (X_N, Cov_N, $\alpha_N^{-1}(0)$, N);
  r6) **if** the check r5) fails then **Reject**;
  r7) **if for each** $i \in \{1, \ldots, n\}$
      prime-attribute (X_N, Cov_N, $\alpha_N^{-1}(i)$, N_i) = Accept,
      where N_i is the i-th child of N in $\mathcal{T}$
    then Accept
    else Reject;
end.

Figure 4: Primality Test.

that the parameter X_N denotes the guess at N. Clearly, $\mathcal{X}$ has to be connected (when considered as a hypergraph with hyperedges $F_{\mathcal{X}}$). For this purpose, we have condition 1.a.

In step 2 we want to guess all FDs $U \to B$ in $F^+_{\mathcal{X}}$ with attributes X_{N_i}, i.e. the intended meaning of Cov_{N_i} is $Cov_{N_i} = F^+_{\mathcal{X}}[X_{N_i}]$. The idea of the ordering that we impose on the elements in $F^+_{\mathcal{X}}$ will become clear when we discuss the check-

primality procedure. The ordering which we are aiming at here is the following:

DEFINITION 5.1. Let $\mathcal{X} \subseteq R$. For $g \in F_{\mathcal{X}}^+$ with $U \to B$, we denote by $\min_{\to}(g)$ the minimal length of a derivation sequence of B from U by means of FDs in $F_{\mathcal{X}}^+$.

We define a partial ordering on $F_{\mathcal{X}}^+$ as follows: Let $g, h \in F_{\mathcal{X}}^+$. If $\min_{\to}(g) < \min_{\to}(h)$, then $g < h$ in $F_{\mathcal{X}}^+$. Moreover, suppose that $g = (U \to B)$ and $h = (V \to B)$, s.t. $U \subset V$ and $\min_{\to}(g) = \min_{\to}(h)$ hold. Then we also set $g < h$ in $F_{\mathcal{X}}^+$.[2] This partial ordering can be extended to a total ordering in an arbitrary way. We shall refer to the resulting ordering as "*FD-ordering*".

Conditions 2.b and 2.c make sure that the guesses at the node N_i are consistent with the guesses at the parent N of N_i. Condition 2.a means that $B \to C$ has to hold for all attributes B, C in the co-antikey $\mathcal{X}$.

The set P_{N_i} computed at step 3 consists of all FDs in Cov_{N_i} for which it has to be verified yet that they are indeed in $F_{\mathcal{X}}^+$. P_{N_i} inherits the ordering from Cov_{N_i} since $P_{N_i} \subseteq Cov_{N_i}$ must hold by condition 3.a. This condition ensures for all FDs $g \in P_N$, that the property $g \in F_{\mathcal{X}}^+$ is verified as long as all attributes occurring in g are contained in $\lambda(N)$.

For $g \in P_{N_i}$, the place where the property $g \in F_{\mathcal{X}}^+$ shall be verified is guessed in step 4, namely: For each $g \in P_{N_i}$, we either verify at node N_i that $g \in F_{\mathcal{X}}^+$ holds (in this case, we guess $\alpha_{N_i}(g) = 0$) or the job to verify this property is passed on to the ℓ-th child of N_i (i.e., we guess $\alpha_{N_i}(g) = \ell$).

The check-primality procedure called at step 5 has two tasks: First, we check (in Check 1) that no attribute of the co-antikey $\mathcal{X}$ is determined by attributes outside $\mathcal{X}$. Second, for all FDs $g \in P_{N_i}$ with $\alpha_{N_i}(g) = 0$, we have to check (in Check 2) that $g \in F_{\mathcal{X}}^+$ indeed holds. Condition (i) means that the FD g is actually represented by $\lambda(N)$. The conditions (ii) and (iii) mean that g follows from one *smaller* FD h or from two smaller FDs h_1, h_2 in Cov_{N_i}. Hence, given that $h \in F_{\mathcal{X}}^+$ or $h_1, h_2 \in F_{\mathcal{X}}^+$, respectively, is checked elsewhere, we may assume at this place that $g \in F_{\mathcal{X}}^+$ holds. It is now also clear why we impose some ordering on $F_{\mathcal{X}}^+$, namely: We have to make sure that there are no cycles in the derivation of the functional dependencies in Cov_{N_i}.

In step 7, the procedure prime-attribute is called recursively for all child nodes M_ℓ of N_i. The first two parameters X_{N_i} and Cov_{N_i} communicate the guesses at N_i to each M_ℓ. The third parameter passes precisely those FDs $g \in P_{N_i}$ to the ℓ-th child for which we have guessed $\alpha_{N_i}(g) = \ell$.

Note that the steps r1 through r7 at the root of $\mathcal{T}$ have exactly the same meaning as the steps 1 through 7 in procedure prime-attribute. Of course, some tasks are slightly simpler since the root N has no parent. Step r1 makes sure that $A \in \mathcal{X}$ indeed holds.

THEOREM 5.2. *Let (R, F) be a relational schema and A an attribute in R. Moreover, let $\mathcal{T}$ be a tree decomposition of (R, F) with width $k \geq 1$. The Primality-Test algorithm in Figure 4 accepts the input $((R, F), \mathcal{T}, A)$, if and only if A is a prime attribute in (R, F).*

[2]Note that for adding attributes to the left-hand side of some FD f, this definition has the following effect: If the additional attributes in $lhs(f)$ allow us to construct a shorter derivation sequence, then the resulting FD is smaller than f. On the other hand, adding to $lhs(f)$ "useless" attributes so to speak produces a greater FD.

Proof. The following equivalence has to be shown: There exists an accepting computation of the non-deterministic Primality-Test algorithm $\Leftrightarrow$ A is prime.

"$\Rightarrow$" We claim that $\mathcal{X} := \bigcup_{N \in \mathcal{T}} X_N$ is a co-antikey in (R, F) with $A \in \mathcal{X}$. Of course, $A \in \mathcal{X}$ by condition r1. By Check 1 in procedure check-primality, the following property is guaranteed: $(\forall f \in F)$ if $rhs(f) \in \mathcal{X}$, then $lhs(f) \cap \mathcal{X} \neq \emptyset$. Hence, it only remains to show that $(B \to C) \in F_{\mathcal{X}}^+$ for all $B, C \in \mathcal{X}$. Actually, by conditions 1.a and 1.b it is guaranteed that the hypergraph produced by the hyperedges X_N (with $N \in \mathcal{T}$) is connected. Hence, it suffices to show that $(B \to C) \in F_{\mathcal{X}}^+$ for all attributes B and C that occur jointly in X_N for some $N \in \mathcal{T}$. By condition 2.a, it is therefore sufficient to prove the following lemma:

LEMMA 5.3. *Let $Cov := \bigcup_{N \in \mathcal{T}} Cov_N$. Then the property $Cov \subseteq F_{\mathcal{X}}^+$ holds.*

Proof Idea for Lemma 5.3. By the conditions 2.b and 2.c and by the connectedness condition in the tree decomposition $\mathcal{T}$, the orderings guessed for the subsets $Cov_N \subseteq Cov$ can be consistently extended to a total ordering on all of Cov. The proof of Lemma 5.3 goes by induction on this ordering. Note that at this point, we do not assume that this ordering is indeed the derivation ordering from Definition 5.1. For Lemma 5.3, any ordering can be used. $\diamond$

"$\Leftarrow$" Suppose that A is prime in (R, F). Hence, there exists a co-antikey $\mathcal{X}$ in (R, F) with $A \in \mathcal{X}$. We construct a computation tree τ of a successful computation of the Primality-Test algorithm as follows: Let N be any node in the tree decomposition $\mathcal{T}$ with $A \in \lambda(N)$. Then we consider $\mathcal{T}$ as rooted at N. The tree structure of the computation tree τ is identical to the tree structure of $\mathcal{T}$. Moreover, our algorithm guesses the following values at each node N: (1) $X_N := \mathcal{X} \cap \lambda(N)$ and (2) $Cov_N := F_{\mathcal{X}}^+[X_N]$, i.e., Cov_N consists of all FDs that are contained in $F_{\mathcal{X}}^+$ and which are made up of attributes in $\lambda(N)$ only. The ordering that we have to guess for Cov_N in step 2 is the FD-ordering on $F_{\mathcal{X}}^+$.

It remains to show that it is possible to guess an appropriate assignment α_N at each node N, s.t. the checks in the check-primality procedure always succeed. Actually, the Check 1 is clearly successful at any node N, i.e., For all attributes B in a co-antikey $\mathcal{X}$, we can be sure that B is not determined by attributes outside $\mathcal{X}$, i.e., for every FD $f \in F$ with $B = rhs(f)$, at least one attribute from $lhs(f)$ must be contained in $\mathcal{X}$.

In order to show that also Check 2 always succeeds for appropriately chosen α_N, the following lemma suffices:

LEMMA 5.4. *Every FD $g \in F_{\mathcal{X}}^+$ added to P_N at some node N by the Primality-Test algorithm will be eventually "eliminated" without producing an error, i.e., f is either eliminated at N (i.e., $\alpha_N(g) = 0$) or g is passed on to some descendant M of N where it is finally eliminated (i.e., $\alpha_M(g) = 0$).*

Proof Idea for Lemma 5.4. The proof is based on an induction on the derivation-ordering. Here we really need precisely the ordering defined in Definition 5.1. $\diamond$

This concludes the proof of Theorem 5.2. $\square$

For the complexity of the Prime-Test algorithm, we have the following upper bound:

THEOREM 5.5. *Let (R, F) be a relational schema whose treewidth is bounded by some constant $k \geq 1$ and let $A \in R$*

be an attribute. It can be decided in linear time whether A is prime in (R, F). Moreover, this decision problem is in LogCFL.

Proof. Note that we do not require in the theorem that a tree decomposition $\mathcal{T}$ of (R, F) with width $k \leq 1$ has to be actually given. However, by [6], a tree decomposition $\mathcal{T}$ can be computed in linear time. Likewise, it has been shown in [15], that the computation of $\mathcal{T}$ is feasible in $\mathbf{L}^{\text{LogCFL}}$ and that $\mathbf{L}^{\text{LogCFL}}(\text{LogCFL}) = \text{LogCFL}$. Hence, in the remainder of the proof, we may assume that $\mathcal{T}$ is given.

The linear time bound is seen as follows: First of all, the data structures guessed in the algorithm Primality-Test depend only on the size k of the labels $\lambda(N)$ and $\lambda(N_i)$ rather than on the size of the entire input. Hence, turning the nondeterministic algorithm into a deterministic one by looping over all possible values of X_{N_i}, Cov_{N_i}, α_{N_i}, etc. increases the time complexity only by a multiplicative constant.

For each collection of guesses of X_{N_i}, Cov_{N_i}, α_{N_i}, etc. the algorithm works by a single traversal of the tree decomposition $\mathcal{T}$, whose size is of course linearly bounded. Moreover, almost all of the steps carried out by the main-program and by the two procedures depend on the labels $\lambda(N)$ and $\lambda(N_i)$ rather than on the size of the entire input. The only two places where one has to be careful are step 7 in the procedure prime-attribute and check 1 in the procedure check-primality, which seem to require linear time whenever they are executed. Note however, that the overall complexity of step 7 (in all of the recursive calls of the procedure prime-attribute) corresponds to the total number of recursive calls of this procedure – which is in fact linear w.r.t. the size of the input. As far as check 1 in the procedure check-primality is concerned, the following slight modification is required: The very purpose of the algorithm of [6] is to construct a tree decomposition $\mathcal{T}$ such that every hyperedge of the hypergraph of (R, F) (or, equivalently, every FD $f \in F$) is covered by at least one bag $\lambda(N)$. This algorithm can be easily extended such that every node N of $\mathcal{T}$ is annotated with the FDs $f \in F$ that N is meant to cover. Moreover, this annotation is done in such a way that each $f \in F$ is used in the annotation of exactly one node N of $\mathcal{T}$. Then check 1 of the procedure check-primality can be modified in that it is only applied to those FDs f which occur in the annotation of the node M of $\mathcal{T}$. The overall complexity of all calls of the procedure check-primality is thus linearly bounded.

Concerning the LogCFL upper bound, it should be noted that the algorithm in Figure 4 can be regarded as a high-level description of an algorithm that is run on an alternating Turing machine (ATM). The existential steps of this ATM are contained in the non-deterministic guesses of this algorithm. The universal steps are encoded by the recursive calls of the prime-attribute procedure.

Recall that ATMs can be characterized in terms of the tree-size (i.e, the number of nodes) of the computation tree τ (where each node corresponds to a configuration of M) and the space required by each configuration. By the characterization of LogCFL in [21], it suffices to show that the size of τ is polynomially bounded and the data manipulated at each node fit into log-space. This is easily verified since the data structures maintained at each node N_i essentially consist of *constantly* many pointers to the input. □

Remark. In Section 8, the analogous fixed-parameter tractability result is shown via an MSO-encoding and by applying Courcelle's Theorem – which makes use of the correspondence between MSO-formulae and *finite tree automata*. In fact, our Primality-Test algorithm in Figure 4 is closely related to a (non-deterministic top-down) finite tree automaton whose states correspond to the (constantly many) possible values of the data structures X_{N_i}, Cov_{N_i}, α_{N_i}, etc. But of course, our dedicated algorithm essentially differs from an FTA that one would obtain via a standard MSO-to-FTA transformation (like the one in [13]).

6. BOUNDED TREEWIDTH & BCNF

Testing whether a relational schema (R, F) is in BCNF is an easy task. In fact, we just have to check that for all FDs $f \in F$, the left-hand side $lhs(f)$ is a key (see [20]). This is clearly feasible in polynomial time. In contrast, when we have already applied some decomposition and if we then want to check whether some subschema $R' \subset R$ is in BCNF, then this problem is NP-complete.

In this section we present an efficient algorithm for the subschema-BCNF problem in case that the set of FDs has bounded treewidth (see Figure 5). The algorithm proceeds by a top-down traversal of the tree decomposition $\mathcal{T}$ rooted at any node. This tree traversal is realized via recursive calls of the procedure sub-not-bcnf. The goal of the algorithm is to find a BCNF-violation in R', i.e., attributes $A, B \in R'$ and a subset $\mathcal{Y} \subseteq R'$, s.t. $\mathcal{Y} \rightarrow_F A$, $A \notin \mathcal{Y}$, and $\mathcal{Y} \not\rightarrow_F B$. The attributes A and B are guessed in step r0. The attribute set $\mathcal{Y} \subseteq R'$ is guessed during the traversal of the tree decomposition $\mathcal{T}$ as follows: At each node N_i in $\mathcal{T}$, we guess Y_{N_i} in step 1 with the intended meaning $Y_{N_i} = \lambda(N_i) \cap \mathcal{Y}$. Condition 1.a guarantees that the guess Y_{N_i} at N_i is consistent with the guess Y_N at the parent N of N_i.

Step 2 aims at guessing the closure $clos_F(\mathcal{Y})$ of $\mathcal{Y}$. The intended meaning of Z_{N_i} is $Z_{N_i} = \lambda(N_i) \cap (clos_F(\mathcal{Y}) \setminus \mathcal{Y})$. The conditions 2.a – 2.d are straightforward. In particular, condition 2.a makes sure that A is indeed determined by $\mathcal{Y}$. As in Section 5, the purpose of the ordering imposed on Z_{N_i} is to prevent circular derivations (cf. Check 3 in the check-not-bcnf procedure). The ordering that we have in mind here is the following:

Definition 6.1. Let $\mathcal{Z} \subseteq clos_F(\mathcal{Y})$ with $\mathcal{Z} \cap \mathcal{Y} = \emptyset$. We define the "*derivation ordering*" on $\mathcal{Z}$ by considering an arbitrary derivation sequence Δ of $\mathcal{Z}$ from $\mathcal{Y}$. Suppose that Δ has the form $\Delta \equiv \mathcal{Y} \rightarrow \mathcal{Y} \cup \{C_1\} \rightarrow \mathcal{Y} \cup \{C_1, C_2\} \rightarrow \ldots \rightarrow \mathcal{Y} \cup \{C_1, C_2, \ldots, C_z\}$, where $\mathcal{Z} = \{C_1, , \ldots, C_z\}$ and $C_i \neq C_j$ if $i \neq j$. Then we set $C_i < C_j$ if $i < j$.[3]

The set O_{N_i} computed at step 3 consists of all attributes in Z_{N_i} for which it has to be verified yet that they are indeed determined by $\mathcal{Y}$. The ordering on O_{N_i} can be taken over from Z_{N_i} since, by condition 3.a, $O_{N_i} \subseteq Z_{N_i}$ holds. The purpose of the assignment function β_{N_i} guessed in step 4 is to determine the place where the property $\mathcal{Y} \rightarrow C$ has to be verified for each $C \in O_{N_i}$: as in the Primality-Test algorithm, 0 means the node N_i itself and $\ell \geq 1$ means the ℓ-th child of N_i.

The check-not-bcnf procedure called at step 5 has the following tasks: Check 1 is clear. Check 2 ultimately guarantees that $(\mathcal{Y} \cup \mathcal{Z}) \supseteq clos_F(\mathcal{Y})$ holds, Check 3 takes care of the property $(\mathcal{Y} \cup \mathcal{Z}) \subseteq clos_F(\mathcal{Y})$.

The remaining steps are clear by the analogy with the Primality-Test algorithm discussed in Section 5. Similarly as Theorem 5.2, the following properties can be shown:

[3]Obviously the derivation-ordering depends on the choice of the derivation sequence Δ. In the sequel, we assume Δ to be arbitrarily chosen but fixed (for any sets $\mathcal{Y}$ and $\mathcal{Z}$).

Algorithm Subschema-Not-BCNF
Input Relational schema (R, F), tree decomposition $\mathcal{T}$ of the primal graph of (R, F) with width k, subset $R' \subseteq R$.
Output "Accept", if A is prime in (R, F).

Procedure check-not-bcnf (Y_M: SetOfAttributes, Z_M,O_M: OrderedSetOfAttributes, M: Node in $\mathcal{T}$)
begin
  Check 1: $A \notin Y_M$, $B \notin Y_M$, and $B \notin Z_M$.
  Check 2: (* $Y_M \cup Z_M$ closed w.r.t. F *)
    $\forall f \in F$: If f is covered by $\lambda(M)$ and
    $lhs(f) \subseteq Y_M \cup Z_M$ then $rhs(f) \in Y_M \cup Z_M$.
  Check 3: $\forall C \in O_M$:
    $\exists (D_1, \ldots, D_m \rightarrow C) \in F$ covered by $\lambda(M)$ s.t.
    $\forall i \leq m$: $(D_i \in Y_M) \vee (D_i \in Z_M \wedge D_i < C)$ holds.
  if all Checks succeed **then Accept**
  else HALT and **Reject**;
end;

Procedure sub-not-bcnf (Y_N: SetOfAttributes, Z_N, O_N: OrderedSetOfAttributes, N_i: Node in $\mathcal{T}$)
begin
1) **Guess** $Y_{N_i} \subseteq \lambda(N_i) \cap R'$ s.t.:
  1.a) $\forall C \in \lambda(N) \cap \lambda(N_i)$: $C \in X_N \Leftrightarrow C \in X_{N_i}$
2) **Guess** ordered set $Z_{N_i} \subseteq \lambda(N_i)$ s.t.:
  2.a) If $A \in \lambda(N_i)$, then $A \in Z_{N_i}$,
  2.b) $Y_{N_i} \cap Z_{N_i} = \emptyset$,
  2.c) $\forall C \in \lambda(N) \cap \lambda(N_i)$: $C \in Z_N \Leftrightarrow C \in Z_{N_i}$
  2.d) The orderings on Z_N and Z_{N_i} are consistent.
3) $O_{N_i} := O_N \cup (Z_{N_i} - Z_N)$;
  3.a) **if not** $O_{N_i} \subseteq Z_{N_i}$ **then HALT** and **Reject**;
4) **Guess** assignment $\beta_{N_i} : O_{N_i} \rightarrow \{0, \ldots, n\}$, where n = number of child nodes of N_i.
5) check-not-bcnf (Y_{N_i}, Z_{N_i}, $\beta_{N_i}^{-1}(0)$, N_i);
6) **if** the check 5) fails then **Reject**;
7) **if for each** $\ell \in \{1, \ldots, n\}$
    sub-not-bcnf (Y_{N_i}, Z_{N_i}, $\beta_{N_i}^{-1}(\ell)$, M_ℓ) = Accept,
    where M_ℓ is the ℓ-th child of N_i in $\mathcal{T}$
  then Accept
  else Reject;
end;

begin (* MAIN *)
  r0) (* initializations *)
    N := root of $\mathcal{T}$;
    Guess $A, B \in R'$ with $A \neq B$;
  r1) **Guess** $Y_N \subseteq \lambda(N) \cap R'$ s.t.:
  r2) **Guess** ordered set $Z_N \subseteq \lambda(N)$ s.t.:
    r1.a) If $A \in \lambda(N)$, then $A \in Z_N$,
    r2.a) $Y_N \cap Z_N = \emptyset$,
  r3) $O_N := Z_N$;
  r4) **Guess** assignment $\beta_N : O_N \rightarrow \{0, \ldots, n\}$, where n = number of child nodes of N.
  r5) check-not-bcnf (Y_N, Z_N, $\beta_N^{-1}(0)$, N);
  r6) **if** the check 5) fails then **Reject**;
  r7) **if for each** $i \in \{1, \ldots, n\}$
    sub-not-bcnf (Y_N, Z_N, $\beta_N^{-1}(i)$, N_i) = Accept,
    where N_i is the i-th child of N in $\mathcal{T}$
    then Accept
    else Reject;
end.

Figure 5: Subschema-Not-BCNF Test.

THEOREM 6.2. *Let (R, F) be a relational schema and let $R' \subseteq R$ be a subschema. Moreover, let $\mathcal{T}$ be a tree decomposition of (R, F) with width $k \geq 1$. The Subschema-Not-BCNF algorithm in Figure 5 accepts input $((R, F), \mathcal{T}, R')$ if and only if R' is not in BCNF.*

Recall from [7] that LOGCFL is closed under complement Hence, in order to establish the LOGCFL-membership of testing whether some subschema R' is in BCNF, it suffices to show that testing whether R' is *not* in BCNF is in LOGCFL. Analogously to Theorem 5.5, we thus get

THEOREM 6.3. *Let (R, F) be a relational schema whose treewidth is bounded by some constant $k \geq 1$ and let $R' \subseteq R$ be a subschema. Then it can be decided in linear time whether R' is in BCNF. Moreover, this decision problem is in* LOGCFL.

7. BOUNDED TREEWIDTH & 3NF

In this section we combine the ideas of the algorithms from the Sections 5 and 6 in order to construct an efficient algorithm also for the primality-test in a subschema and ultimately for the 3NF-test of a subschema.

Given a relation (R,F), a subschema $R' \subseteq R$, an attribute $A \in R'$ and a tree decomposition $\mathcal{T}$ of (R, F), s.t. the width of $\mathcal{T}$ is bounded by some constant $k \geq 1$, the primality of A in the subschema $(R', F[R'])$ can be tested as follows. It is convenient to refer to the Primality-Test algorithm as PT and to the Subschema-Not-BCNF algorithm as SNB:

Initializations. As in PT, we search for some node N in $\mathcal{T}$ with $A \in \lambda(N)$ and consider $\mathcal{T}$ as rooted at N.

Data structures. At every node N_i in $\mathcal{T}$, we need exactly the same data structures as in PT, namely: X_{N_i}, Cov_{N_i}, and α_{N_i} with the analogous meaning as before. In particular, $\mathcal{X} := \bigcup_{N \in \mathcal{T}} X_N$ will ultimately be a co-antikey in $(R', F[R'])$ with $A \in \mathcal{X}$. In addition, we also need the data structures Y_{N_i}, Z_{N_i}, and β_{N_i} from SNB with the same meaning as before. In particular, $\mathcal{Y} := \bigcup_{N \in \mathcal{T}} Y_N$ is a subset of R' and $\mathcal{Z} := \bigcup_{N \in \mathcal{T}} Z_N = clos_F(\mathcal{Y}) \setminus \mathcal{Y}$. Only the variables A and B from SNB are omitted here.

Guesses. The data structures X_{N_i}, Cov_{N_i}, and α_{N_i} are maintained in the same way as described in steps 1 through 7 in PT. For the data structures Y_{N_i}, Z_{N_i}, and β_{N_i} from SNB, we need one significant modification: Rather than guessing Y_{N_i} at each node (see step 1 in SNB), we set $Y_{N_i} := (R' \setminus X_{N_i}) \cap \lambda(N_i)$. In other words, $\mathcal{Y} := \bigcup_{N \in \mathcal{T}} Y_N$ will ultimately yield $R' \setminus \mathcal{X}$.

Checks. The Check 1 in SNB is dropped (since we are not dealing with A and B from SNB). But Check 2 from SNB is taken over unchanged. Likewise, the Checks 2 and 3 from PT can be left unchanged. But Check 1 from PT has to be replaced by the condition $A \notin Z_M$. In other words, we have to make sure that A is not determined by the complement $\mathcal{Y}$ of the co-antikey $\mathcal{X}$ (via the FDs in F).

By combining the results from the Sections 5 and 6, we thus get the following theorem:

THEOREM 7.1. *Let (R, F) be a relational schema whose treewidth is bounded by some constant $k \geq 1$ and let $R' \subseteq R$ be a subschema, and $A \in R'$ an attribute. Then it can be decided in linear time whether A is prime in $(R', F[R'])$. Moreover, this decision problem is in* LOGCFL.

Finally, we describe an algorithm for testing whether some subschema $R' \subseteq R$ is *not* in 3NF. Suppose that we are given a relational schema (R, F), a subschema $R' \subseteq R$, and a tree decomposition $\mathcal{T}$ of (R, F), s.t. the width of $\mathcal{T}$ is bounded by some constant $k \geq 1$. Our "Subschema-Not-3NF" algorithm is obtained by the following modification of our Subschema-Not-BCNF algorithm: When guessing the attribute A in step r0, we have to check that A is *not prime* in $(R', F[F'])$. By Theorem 7.1 and by the fact that LOGCFL is closed under complement (see [7]), this check can be done by a

LogCFL-algorithm. But then, since $\mathbf{L}^{\text{LogCFL}}(\text{LogCFL}) = \text{LogCFL}$ (see [15]), the whole test that $R' \subseteq R$ is *not* in 3NF, is in LogCFL. Making once more use of the closure of LogCFL under complement, we thus get

THEOREM 7.2. *Let (R, F) be a relational schema whose treewidth is bounded by some constant $k \geq 1$ and let $R' \subseteq R$ be a subschema. It can be decided in linear time whether R' is in 3NF. Moreover, this decision problem is in* LogCFL.

8. INCIDENCE GRAPH

In this section, we first encode the six decision problems considered so far (i.e., PRIMALITY, 3NFTEST, and BCNFTEST – both for a schema and for a subschema) with Monadic Second-Order (MSO) formulae. Then we show that if the treewidth of the incidence graph of the relational schema is bounded, the fixed-parameter tractability results can be established using Courcelle's Theorem.

THEOREM 8.1 ([8]). *Let φ be a fixed MSO-sentence and k a fixed constant. Deciding whether φ holds for an input graph G (more generally, for an input structure $\mathfrak{A}$) can be done in linear time if the treewidth of the graphs (resp. of the structures) under consideration is bounded by k.*

Due to the well-known analogy between functional dependencies and propositional Horn clauses, we use a logical notation to facilitate the presentation. Let (R, F) be a relational schema, an *interpretation* of F is simply a subset X where $X \subseteq R$. If X satisfies F, then X is called a *model* of F, written as $X \models F$.

Let (R, F) be a relational schema, and $\mathcal{H}$ be the hypergraph of (R, F) (see Definition 2.2). We denote the treewidth of the *incidence graph* of (R, F) as $tw_I(\mathcal{H})$. More specifically, we can represent (R, F) by a relational structure $\mathfrak{A}(R, F)$ based on the relations $attr(.), rule(.), Head(.,.)$ and $Tail(.,.)$ with the following intended meaning: $rule(h)$ means that h is an FD rule in F, and $attr(a)$ means that a is an attribute in R; $Head(x, h)$ (resp. $Tail(x, h)$) means that x occurs at the right-hand side (resp. left-hand side) in the FD rule h. Let $\mathcal{H}_{\mathfrak{A}}$ be the hypergraph of $\mathfrak{A}(R, F)$ (see Definition 2.3), It is obvious that $tw_I(\mathcal{H}) = tw_I(\mathcal{H}_{\mathfrak{A}})$.

We first encode the property $X \models F$.

MSO encoding of $X \models F$ (with relational schema (R, F) and $X \subseteq R$)

$$(\forall h) rule(h) \rightarrow (\exists a)[(Head(a, h) \wedge a \in X) \vee (Tail(a, h) \wedge a \notin X)]$$

Next we introduce the MSO formulae encoding superkey, key and some auxiliary predicates.

MSO encodings of superkey, key and some auxiliary predicates

$$X \subseteq R \equiv (\forall a) a \in X \rightarrow attr(a)$$

$$X \subset R \equiv X \subseteq R \wedge (\exists a)(attr(a) \wedge a \notin X)$$

$$Clo(X, F, Y) \equiv X \subseteq Y \wedge Y \models F \wedge (\forall Y')[(X \subseteq Y' \wedge Y' \subset Y) \rightarrow \neg(Y' \models F)]$$

$$SK(X, F, R) \equiv Clo(X, F, R)$$

$$K(X, F, R) \equiv SK(X, F, R) \wedge \neg(\exists X')[X' \subset X \wedge SK(X', F, R)]$$

$$Lhs(X, h) \equiv (\forall a) a \in X \rightarrow Tail(a, h)$$

The predicates defined above have the following meaning:
$Clo(X, F, Y)$: Y is the closure of X w.r.t. F;
$SK(X, F, R)$: X is a superkey of the schema (R, F);
$K(X, F, R)$: X is a key of the relational schema (R, F);
$Lhs(X, h)$: X is contained in the lhs of the FD rule h.

With the above auxiliary predicates, the following MSO encoding is straightforward.

MSO encodings of PRIMALITY, BCNF and 3NF

$$Prime(a, F, R) \equiv (\exists X)[K(X, F, R) \wedge a \in X]$$

$$BCNF(F, R) \equiv (\forall h) rule(h) \rightarrow (\exists X)[Lhs(X, h) \wedge SK(X, F, R)]$$

$$3NF(F, R) \equiv (\forall h) rule(h) \rightarrow [(\exists X)[Lhs(X, h) \wedge SK(X, F, R)] \vee (\exists b)[Head(b, h) \wedge Prime(b, F, R)]]$$

Now we consider the problems PRIMALITY, BCNF, and 3NF for a subschema. Let (R, F) be the relational schema and $(R', F[R'])$ be the *subschema* of (R, F), where $R' \subseteq R$. We construct a relational structure $\mathfrak{A}'(R, F, R')$, which contains $\mathfrak{A}(R, F)$ and a new relation $attr'(.)$, where $attr'(a)$ means that a is an attribute in R'. Let $\mathcal{H}_{\mathfrak{A}'}$ be the hypergraph of $\mathfrak{A}'(R, F, R')$. Because $attr'$ is a unary relation and does not affect the treewidth of the incidence graph, we have $tw_I(\mathcal{H}_{\mathfrak{A}'}) = tw_I(\mathcal{H}_{\mathfrak{A}})$. Moreover, auxiliary predicates concerning R', such as $X \subseteq R'$, can be easily constructed by replacing $attr$ with $attr'$ from the predicates concerning R.

MSO encodings of PRIMALITY, BCNF and 3NF with subschema

$$PrimeS(a, F, R, R') \equiv \exists X \; K(X, F, R') \wedge a \in X$$

$$BCNFS(F, R, R') \equiv (\forall X, a)[(\exists Y)[Clo(X, F, Y) \wedge a \in Y] \wedge X \subseteq R' \wedge a \notin X \rightarrow SK(X, F, R')]$$

$$3NFS(F, R, R') \equiv (\forall X, a)[(\exists Y)[Clo(X, F, Y) \wedge a \in Y] \wedge X \subseteq R' \wedge a \notin X \rightarrow [SK(X, F, R') \vee Prime(a, F, R')]]$$

Since all the decision problems considered in this paper can be expressed by means of MSO sentences, the following fixed-parameter tractability result is an immediate consequence of Courcelle's Theorem (Theorem 8.1).

THEOREM 8.2. *The decision problems PRIMALITY, 3NFTEST, and BCNFTEST (both for a schema and for a subschema) can be solved in linear time, if the incidence graph of (R, F) has bounded treewidth.*

9. NF-DECOMPOSITION REVISITED

The intractability of the decision problems recalled in the introduction is a severe obstacle to satisfactory normal form decomposition algorithms for 3NF and BCNF. In particular, as was illustrated in Example 1.1, without the ability to recognize the normal form, one inevitably runs the risk of further decomposing a schema even though the desired normal form has already been reached. However, in many situations, the relational schema under investigation has low treewidth. In this case, our algorithms presented in the previous sections open the grounds for a completely new approach to normal form decomposition. Rather than starting from the FDs and defining subschemas bottom-up so to speak, one can start from the schema itself, check for normal form violations and define appropriate subschemas top-down. In this section we present a simple 3NF decomposition algorithm based on this new approach, see Figure 6.

The 3NF-Decomposition algorithm obviously computes a lossless join decomposition into 3NF subschemas. Moreover, by Theorem 7.2, it works in polynomial time (note that our NF-algorithms from Sections 6 and 7 can be easily extended so as to output a concrete NF-violation rather than just answering "Accept" or "Reject"). Of course, there is ample space for improvements and extensions of this algorithm. We conclude this section by outlining just a few directions of further refining this algorithm and for extending it to a BCNF decomposition algorithm:

Algorithm 3NF Decomposition
Input Relational schema (R, F), tree-decomposition $\mathcal{T}$ of the primal graph of (R, F) with width k.
Output Set $\mathcal{S}_{3NF}$ of subschemas in 3NF.

begin
  $\mathcal{S}_{3NF} := \emptyset$; $R' := R$; stop := false;
  repeat
    if R' contains a 3NF-violation $A_1, \ldots, A_m \to B$ **then**
      $\mathcal{S}_{3NF} := \mathcal{S}_{3NF} \cup \{\{A_1, \ldots, A_m, B\}\}$;
      $R' := R' \setminus \{B\}$;
    else
      $\mathcal{S}_{3NF} := \mathcal{S}_{3NF} \cup \{R'\}$;
      stop := true;
    fi;
  until stop;
end;

Figure 6: 3NF-Decomposition Algorithm.

(1) One is normally interested in an *FD-preserving* decomposition. Hence, as a post-processing step, one should check for all FDs in F whether they are embedded in the resulting set of subschemas. For every FD $A_1, \ldots, A_m \to B$ not embedded in $\mathcal{S}_{3NF}$, we can simply add the subschema $\{A_1, \ldots, A_m, B\}$ to $\mathcal{S}_{3NF}$. All this can be done in polynomial time (see [3]).

(2) When a 3NF-violation $A_1, \ldots, A_m \to B$ has been detected, one should not only split off the single attribute B from R'. Instead, one should search for further attributes $B_1, \ldots, B_\ell$ which are also determined by $A_1, \ldots, A_m$ and split off all these attributes together. Actually, the resulting subschema $S = \{A_1, \ldots, A_m, B, B_1, \ldots, B_\ell\}$ is not necessarily in 3NF. But this is no problem. We just have to check whether the subschema S already is in 3NF and, otherwise, apply the 3NF decomposition recursively to this subschema. By Theorem 7.2, also the 3NF-test in a subschema S of (R, F) can be done efficiently in case of bounded treewidth.

(3) Similarly, when we use the idea of the 3NF-Decomposition algorithm for a BCNF decomposition, then the subschemas produced are not necessarily in BCNF. But again, this can be efficiently detected (see Theorem 6.3) and we just have to apply the BCNF decomposition recursively to the subschema. Unfortunately, there is no way to guarantee that we actually find an *FD-preserving* decomposition into BCNF. However, by results shown in [2] (a lossless join, FD-preserving BCNF decomposition does not necessarily exist and it is coNP-hard to decide whether one exists), this can hardly be helped.

10. CONCLUSION

In this paper, we have presented new algorithms for six fundamental decision problems in database design, namely: PRIMALITY, 3NFTEST, and BCNFTEST (both for a schema and for a subschema). These algorithms work in linear time in case that the input relational schema has bounded treewidth. Along the way to these new algorithms, we have provided a new characterization of primality and a heuristics for dealing with keys.

To the best of our knowledge, these are the first results on tractable fragments of decision problems in database design via bounded treewidth. As we have already mentioned before, we are convinced that most relational schemas encountered in practice tend to have low treewidth. Hence, it is clearly worthwhile to further investigate the potential benefit of the concept of treewidth in this area.

The algorithms presented in this work are based on the definition of treewidth via the primal graph of a hypergraph H. For the case that the treewidth is defined via the incidence graph, we have proved the corresponding fixed-parameter tractability results via appropriate MSO encodings. Dedicated algorithms also in the latter case are left for future work. Likewise, applying other notions of treewidth (like directed treewidth) to the problems studied here is an interesting target for future research.

Problems related to the database design problems studied here also arise in Artificial Intelligence. In particular, Abduction (which is an important diagnosis technique) is strongly related to the PRIMALITY problem. Unfortunately, as was shown in [10], the major decision and computation problems of Abduction are NP-hard, even if a system is described by propositional Horn clauses only. In a forthcoming paper, we shall point out how the tractability results obtained here can be fruitfully applied to Abduction and other reasoning problems in Artificial Intelligence.

11. REFERENCES

[1] W. Armstrong. "Dependency Structures of Data Base Relationships". In *IFIP Congress*, pages 580–583, 1974.

[2] C. Beeri and P. A. Bernstein. "Computational Problems Related to the Design of Normal Form Relational Schemas". *ACM Trans. Database Syst.*, 4(1):30–59, 1979.

[3] C. Beeri and P. Honeyman. "Preserving Functional Dependencies". *SIAM J. Comput.*, 10(3):647–656, 1981.

[4] C. Berge. *Hypergraphs*. North Holland, Amsterdam, 1989.

[5] P. A. Bernstein. "Synthesizing Third Normal Form Relations from Functional Dependencies". *ACM Trans. Database Syst.*, 1(4):277–298, 1976.

[6] H. L. Bodlaender. "A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth". *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[7] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. "Two Applications of Inductive Counting for Complementation Problems". *SIAM J. Comput.*, 18(3):559–578, 1989.

[8] B. Courcelle. "Graph Rewriting: An Algebraic and Logic Approach". In *Handbook of Theoretical Computer Science, Volume B*, pages 193–242, 1990.

[9] J. Demetrovics and V. D. Thi. "Keys, Antikeys and Prime Attributes". *Annales Univ. Sci. Budapest*, Sect. Comp.(8):35–52, 1987.

[10] T. Eiter and G. Gottlob. "The Complexity of Logic-based Abduction". *J. ACM*, 42(1):3–42, 1995.

[11] T. Eiter and G. Gottlob. "Identifying the Minimal Transversals of a Hypergraph and Related Problems". *SIAM J. Comput.*, 24(6):1278–1304, 1995.

[12] T. Eiter, G. Gottlob, and K. Makino. "New Results on Monotone Dualization and Generating Hypergraph Transversals". In *STOC*, pages 14–22, 2002.

[13] J. Flum, M. Frick, and M. Grohe. "Query Evaluation via Tree-decompositions". *J. ACM*, 49(6):716–752, 2002.

[14] M. L. Fredman and L. Khachiyan. "On the Complexity of Dualization of Monotone Disjunctive Normal Forms". *J. Algorithms*, 21(3):618–628, 1996.

[15] G. Gottlob, N. Leone, and F. Scarcello. "Computing LOGCFL Certificates". *Theor. Comput. Sci.*, 270(1-2):761–777, 2002.

[16] G. Gottlob, N. Leone, and F. Scarcello. "Hypertree Decompositions and Tractable Queries". *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

[17] C. L. Lucchesi and S. L. Osborn. "Candidate Keys for Relations". *J. Comput. Syst. Sci.*, 17(2):270–279, 1978.

[18] H. Mannila and K.-J. Räihä;. *The Design of Relational Databases*. Addison-Wesley, 1992.

[19] K. K. Nambiar, B. Gopinath, T. Nagaraj, and Manjunath. "Boyce-Codd Normal Form Decomposition". *J. Comp. and Math. with Appl.*, 33(4), 1997.

[20] S. L. Osborn. "Testing for Existence of a Covering Boyce-Codd Normal Form". *Inf. Process. Lett.*, 8(1):11–14, 1979.

[21] W. L. Ruzzo. "Tree-size Bounded Alternation". *J. Comput. Syst. Sci.*, 21(2):218–235, 1980.

[22] R. Tarjan. "Depth-first Search and Linear Graph Algorithms". *SIAM Journal on Computing*, 1(2):146–160, June 1972.