An Efficient Algorithm to Compute the Candidate Keys of a Relational Database Schema

HOSSEIN SAIEDIAN AND THOMAS SPENCER

Department of Computer Science, University of Nebraska at Omaha, Omaha, NE 68182, USA
email: hossein@cs.unomaha.edu

We provide an efficient algorithm for computing the candidate keys of a relational database schema. The algorithm exploits the 'arrangement' of attributes in the functional dependencies to determine which attributes are essential and useful for determining the keys and which attributes should not be considered. A more generalized algorithm using *attribute graphs* is then provided which allows a uniform and simplified solution to find all possible keys of a relational database schema when the attribute graph of Functional Dependencies (FDs) is not strongly connected.

Received August 1993; revised February 1996

1. BACKGROUND

A relational database schema R, denoted by $R(A_1, A_2, \ldots, A_n)$, is a set of attributes. An instance of R, denoted by r, is a subset of the Cartesian product of the domains of attributes of R, i.e. $r(R) \subseteq \operatorname{dom} A_1 \times \operatorname{dom} A_2 \times \ldots \times \operatorname{dom} A_n$. The elements of a relation are referred to as tuples. A *key* of relation r is a subset of attributes of R with the following two properties:

- *Uniqueness.* No two distinct tuples of r have the same values for the key attributes. Thus if K denotes a key of r, then for any two distinct tuples μ and ν of r, $\mu[K] \neq \nu[K]$.
- *Minimality.* No proper subset of K should have the above property.

K is considered a *superkey* if it satisfies the uniqueness but not the minimality property. Those attributes of R that participate in a key are called *prime* attributes. If a relation has more than one key, each key is referred to as a *candidate* key of R.

Functional dependencies (FDs) represent the inter-relationship among attributes of a relation. A functional dependency is denoted by $X \rightarrow Y$ where $X \subseteq R$ and $Y \subseteq R$, and is read 'X functionally determines Y.' Such a dependency specifies the following constraint on the current value of r: if $\forall \mu, \nu \in r, \mu[X] = \nu[X]$ then $\mu[Y] = \nu[Y]$.

Given a set of FDs Γ for R, Γ is said to *logically imply* $X \rightarrow Y$ (written as $\Gamma \models X \rightarrow Y$) if every instance r of R that satisfies the dependencies in Γ also satisfies $X \rightarrow Y$. The *closure* of Γ, denoted by Γ^+, is a set of all FDs that are logically implied by Γ, i.e. $\Gamma^+ = \{X \rightarrow Y | \Gamma \models X \rightarrow Y\}$. The *inference* rules [1] (also in Ullman [2, pp. 384–385]) can be used to infer all FDs of Γ^+.

The inferences rules are as follows [2, pp. 384–385]:

- If $Y \subseteq X \subseteq R$ then $X \rightarrow Y$,
- $X \rightarrow Y$, then $XZ \rightarrow YZ$ where $Z \subseteq R$,
- $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$,
- $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$,
- If $X \rightarrow Y$ and $Z \subseteq Y$, then $X \rightarrow Z$, and
- $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$.

A more formal definition of the key for a relation R can now be given. Given a relation R with attributes $A_1, A_2, \ldots, A_n$, and a set of FDs Γ, $K \subseteq R$, is a key of R if:

- $K \rightarrow A_1 A_2 \ldots A_n \in \Gamma^+$ and
- For no $Y, Y \subset K$, is $Y \rightarrow A_1 A_2 \ldots A_n \in \Gamma^+$.

In addition to computing the closure of a set of FDs Γ, inference rules can be employed to compute the closure of a set of attributes X. The closure of a set of attributes X, written as X^+, is the set of attributes A that are functionally determined by X, that is, $X \rightarrow A$ can be deduced from Γ via inference rules. X^+ is called the closure of X under Γ.

2. COMPUTING THE CANDIDATE KEYS OF *R*

A universal relation is a relation which includes all attributes of the database. Given a universal relation scheme, R, and a set of functional dependencies, Γ, it is essential to determine correctly all candidate keys of R. Most database authors provide a definition for the key, but no algorithm for computing it. Others, e.g. Maier [3] and Ullman [2], provide algorithms for computing the closure of a set of attributes (or a set of FDs) but the calculation of a key is left to the readers using the closure algorithms. Determining the candidate keys for a small relation schema with a small set of FDs may be trivial but, if a relation has a relatively large number of attributes and/or FDs, then determining the keys may not be a trivial process. Lucchesi and Osborn [4] designed an algorithm which finds all the keys and whose running time is a polynomial of the size of the input and the size of the output. Specifically, their algorithm runs in $O(FKA(K + A))$, where A is the number of attributes, F is the number of functional dependencies, and K is the number of keys returned. We will re-visit this algorithm in Section 4.

Elmasri and Navathe [5] offer the following algorithm:

```
set $K \leftarrow R$;
for each attribute $A \in K$
```

```
compute $\{K - A\}^+$ with respect to $\Gamma$
if $\{K - A\}^+$ contains all attributes of $R$ then
  set $K \leftarrow K - \{A\}$
```

The above algorithm has one major deficiency: it returns only one key for R, and the returned key depends on the order in which attributes were removed. For example, for $R(ABCDEF)$ with a set of FDs Γ, if we start removing attributes from the right side, that is, F, followed by E, followed by D, we may conclude that ABC is a key, while not realizing that, for example, E by itself, or F by itself, or a combination of EF, or E or F combined with any of the attributes A, B or C are also keys of R. An alternative algorithm for finding a single key is given by Kundu [6]. (We will apply our algorithm to one of the examples in [6] in Section 3 and invite interested readers to compare the two solutions.) For some rare applications, it *may* be enough to find only one candidate key and thus does not matter which key is found. The problem of finding a candidate key is much easier than finding all the candidate keys. In practice, it is important to find all the candidate keys. Our algorithm will find not one but all the candidate keys.

Using the closure algorithms while not considering the arrangement of attributes in Γ may be time consuming and inefficient. Some attributes may never participate in a key. Consideration of other attributes may lead to superkeys while late consideration of certain other attributes may simply prolong the process or lead to incorrect answers.

We believe that a certain *categorization* of attributes (based on their appearance on the left-hand side and right-hand side of FDs) could expedite the process substantially and lead to correct solutions, perhaps early in the process. This categorization includes determining those attributes that must be part of a key those attributes that will not be part of any key and those attributes that may be part of the key. The algorithm is described in Section 3. Examples are provided to illustrate the algorithm.

We will introduce a generalized algorithm in Section 4 which uses *attribute* graphs to represent data dependencies. This algorithm provides a uniform and simplified solution to find all possible keys of a relational database scheme when the attribute graph of FDs is not *strongly connected*. We note here that graph algorithms have been used in literature to represent and manipulate functional dependencies in relational database schemas. Of interest is an article by Ausiello, D'atri and Saccà [7] who provide an approach for homogeneous treatment of several related aspects of relational database schemas, namely closure, minimalization and synthesizing a relational scheme in 3NF. But key finding issues are not treated there. Yet another related article by Biskup *et al*. [8] explores the relational database schemes having a unique minimal key but the emphasis is on their relationship with normal form relation schemes and the lattice theoretic issues.

3. THE KEY FINDING ALGORITHM

We assume a relation scheme R and a *minimal* set of FDs Γ. (Since the term 'minimal' has a different meaning in the literature, some database authors, e.g. Maier [9], refer to a minimal set of FDs as a *canonical* set of FDs.) A set of functional dependencies Γ is minimal (or canonical) if it has the following properties [2, pp. 390–391]:

1. The right side of every FD contains a single attribute.
2. There is no extraneous attribute on the left side of any FD, i.e. for no $X \to A$ in Γ, and $Y \subset X$ is $\Gamma - \{X \to A\} \cup \{Y \to A\}$ equivalent to Γ.
3. There are no redundant FDs in Γ, i.e. for no $X \to A$ in Γ is the set $\Gamma - \{X \to A\}$ equivalent to Γ.

The following three steps are used to find the candidate keys of a relation scheme.

Step 1: Determining $\mathcal{L}, \mathcal{B}$ and $\mathcal{R}$

Given a relation R and a set of FDs Γ, divide the attributes of R into three distinct sets $\mathcal{L}, \mathcal{R}$ and $\mathcal{B}$. The set $\mathcal{L}$ contains those attributes of R that occur only on the left-hand side of some FDs in Γ. Similarly, the set $\mathcal{R}$ represents those attributes that occur only on the right-hand side of some FDs in Γ, while $\mathcal{B}$ is the set representing those attributes that occur on both sides of some FDs in Γ. Observe that $\mathcal{L} \cap \mathcal{R} = \emptyset$ and $\mathcal{L} \cap \mathcal{B} = \emptyset$ and $\mathcal{R} \cap \mathcal{B} = \emptyset$. Furthermore, we assume throughout this article that $\mathcal{L} \cup \mathcal{R} \cup \mathcal{B} = R$.

Step 2: Considering $\mathcal{L}$

Consider the set $\mathcal{L}$. If $\mathcal{L}$ is not empty, then all attributes participating in $\mathcal{L}$ are prime attributes:

LEMMA 1. *For every attribute $A \in R$, if $A \in \mathcal{L}$, then A must be part of every candidate key of R.*

Proof 1. Suppose K is a candidate key of R and $A \notin K$. By definition $K \to A \in \Gamma^+$. This implies that there must be an FD $X \to A$. However $X \to A$ contradicts $A \in \mathcal{L}$. Thus A must be a part of K. □

We begin our process by computing the closure of attributes in $\mathcal{L}$. If attributes contained in $\mathcal{L}$ form a key for R, then $\mathcal{L}$ will be the only key of R:

LEMMA 2. *If $\mathcal{L}^+ = R$ under Γ, then $\mathcal{L}$ forms the only key of R. (In other words, if $\mathcal{L} \to R \in \Gamma^+$, then $\mathcal{L}$ would be the only key of R.)*

Proof 2. Let K be a key of R. By Lemma 1 $\mathcal{L} \subseteq K$. Since $\mathcal{L}^+ = R$, if $\mathcal{L} \subset K$, then K would be a superkey. Therefore $\mathcal{L} = K$. □

If a key is found, stop. Otherwise proceed to Step 3.

Step 3: Considering $\mathcal{B}$

If the set $\mathcal{L}$ does not produce a key for R in Step 2, then begin adding attributes, one by one, from the set denoted by $\mathcal{B}$ to attributes of $\mathcal{L}$ and compute their closure. Attributes should be added to $\mathcal{L}$ in turn to ensure that all candidate keys of R are found. (If $\mathcal{L}$ is empty, then begin by computing the closure of attributes in the set $\mathcal{B}$.)

Please note that since we are only interested in computing the candidate keys of a universal relation, attributes in $\mathcal{R}$ need not be considered as they will never end up in a key as shown in the following lemma:

Lemma 3. *For every attribute $A \in R$, if $A \in \mathcal{R}$, then A will not be part of any candidate key of R.*

Proof 3. Suppose K is a key of R. Furthermore, suppose there exists an attribute $A \in \mathcal{R}$. (Thus, $A \notin \mathcal{L}$ and $A \notin \mathcal{B}$.) Assume A is a part of K. Let $X = (K - A)^+$. Since K is key of R, $X \subset R$. Let $Y = R - X$. Also, since K is a key of R, $K \rightarrow Y$. There must exist an FD $AW \rightarrow V$, where $V \subseteq Y$ and W may or may not be empty. But $AW \rightarrow V$ contradicts $A \in \mathcal{R}$. Therefore A cannot be a part of K. □

Example 1

Consider the relation schema $U(CTHRSG)$ introduced by Ullman [2, pp. 407] to represent a small database in a university, where $C =$ course, $T =$ teacher, $H =$ hour, $R =$ room, $S =$ student and $G =$ grade, and the following functional dependencies Γ:

1. $C \rightarrow T$, i.e. each course has one teacher.
2. $HR \rightarrow C$, i.e. only one course meets in a room at one time.
3. $HT \rightarrow R$, i.e. a teacher is in at most one room at one time.
4. $CS \rightarrow G$, i.e. each students receives one grade in each course.
5. $HS \rightarrow R$, i.e. a student can be in only one room at a given time.

Solution

Step 1: Determine $\mathcal{L}, \mathcal{B}$, and $\mathcal{R}$: $\mathcal{L} = \{HS\}, \mathcal{B} = \{CRT\}$, and $\mathcal{R} = \{G\}$. According to Lemma 1, H and S must be part of any key while, according to Lemma 3, G will not participate in any key. Note that according to the algorithm,

- we need not compute the closure of any single attribute because $\mathcal{L}$ contains two attributes (namely H and S),
- we need not consider attribute G because it appears in $\mathcal{R}$, and
- we should consider attributes of $\mathcal{L}$ (namely, H and S) before considering any other combination of attributes.

Step 2: Considering $\mathcal{L}$: $\mathcal{L}^+ = \{HS\}^+ = HSRCTG\ldots$
Thus $\{HS\}$ is a key of U. According to Lemma 2, $\{HS\}$ is the *only* key of U. We need not consider any other combination of attributes.

Note that in Example 1, we considered a schema for which the set $\mathcal{L}$ was non-empty and formed the only key for the schema. In other words, we only had to consider $\mathcal{L}$ (i.e. step 2 of the algorithm). For the next example, we will consider a schema for which $\mathcal{L}$ will be non-empty but will not form a key.

Example 2

We now consider one of the examples of Kundu [6]. Note that our algorithm will find *all* of the candidate keys. A relational schema $S(ABCDEF)$ with the following set of functional dependencies is assumed: $\Gamma = \{AD \rightarrow B, AB \rightarrow E, C \rightarrow D, B \rightarrow C, AC \rightarrow F\}$.

Solution

Step 1: Determine $\mathcal{L}, \mathcal{B}$, and $\mathcal{R}$: $\mathcal{L} = \{A\}, \mathcal{B} = \{BCD\}$, $\mathcal{R} = \{EF\}$. According to Lemma 1, A must be a part of every candidate key of S while, according to Lemma 3, E and F may never participate in any. We begin by computing the closure of $\mathcal{L}$ in Step 2.

Step 2: Consider $\mathcal{L}$ only: $\mathcal{L}^+ = A^+ = A$. Thus A is not a key of S.

Step 3: Consider $\mathcal{B}$: We now consider adding attributes from $\mathcal{B}$ to $\mathcal{L}$ and compute their closure to find all candidate keys:

$$\{AB\}^+ = ABECDF. \quad AB \text{ is a key of } S.$$
$$\{AC\}^+ = ACDB\ldots \quad AC \text{ is a key of } S.$$
$$\{AD\}^+ = ADB\ldots \quad AD \text{ is a key of } S.$$

Thus, AB, AC and AD are the only candidate keys of S. No other single attribute or combination of attributes needs to examined. □

Brief Discussion. According to our algorithm, AB, AC and AD are the only keys of S. Note that in order to find the key(s) of S:

- we did not need to compute the closure of any single attribute other than A,
- we did not need to consider any combination such as BC, BD, DC, BCD, etc. because A must be in every candidate of S,
- we did not need to consider any combination of A with either E or F because these two attributes may never be in any key of S, and
- other combinations, e.g. ABC, will yield a superkey.

If $\mathcal{L} = \emptyset$ and $\mathcal{R} \neq \emptyset$, it would imply that except for attributes of $\mathcal{R}$, every other attribute has a fair chance of being part of a candidate key. If $\mathcal{L} = \emptyset$ and $\mathcal{R} = \emptyset$ (that is, $\mathcal{B}$ contains all attributes), then it implies the worst case scenario, i.e. every attribute might be a potential component of a key. One solution is to use the existing approaches. We provide a better approach in the following section.

4. A MORE GENERAL HEURISTIC

Even if $\mathcal{B}$ contains all the attributes, it is still often possible to generalize these ideas to obtain an algorithm that is substantially better than the brute force algorithm of trying all possible sets of attributes to find the keys. To define this algorithm, we need some additional notation.

Definition 1. Let R be the relational database scheme we want to compute the keys of, and let $\mathcal{A}$ be the set of attributes of R. Then, if $\mathcal{B} \subseteq \mathcal{A}$, define $\bar{\mathcal{B}} = \mathcal{A} - \mathcal{B}$.

We will have occasion to talk about several different relational database schemes with attributes that are subsets of $\mathcal{A}$. Thus, if we make a statement like $X \rightarrow Y$, it matters which relational database scheme we are talking about, since the different relational database schemes may have a totally different set of functional dependencies.

Definition 2. We write $(X \rightarrow Y)_{R'}$ to mean that X functionally determines Y in the relational database scheme R'. We only write $X \rightarrow Y$ if it is clear which relational database scheme we are talking about. Similarly, we write $(X^+)_{R'} = Y$ to mean that the closure of X in the relational database scheme R' is Y.

The final tool that we need is the attribute graph of a relational database scheme.

Definition 3. The attribute graph $G[R]$ of the relational database scheme R is a directed graph with one vertex for each attribute of R. There is an edge from A to B if and only if A is on the left side of some functional dependency and B is on the right side of the same functional dependency.

If $G[R]$ is not strongly connected, then the problem of finding the keys for R splits into the problems of finding the keys of two smaller relational database schemes R_c and R_r containing disjoint subsets of the attributes of R. Consider the relational database scheme with attributes $\{A, B, C, D, E, F\}$ and functional dependencies $\{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow D, D \rightarrow C, BD \rightarrow E, E \rightarrow F, F \rightarrow E\}$. Its attribute graph is shown in Figure 1. To see how R_c and R_r are defined, we need some concepts from graph theory.

Definition 4. Two vertices u and v in a directed graph G are in the same strongly connected component *if* and *only if* there is a path from u to v and from v to u.

Definition 5. A strongly connected component C of a directed graph G is a source component *if all edges to a vertex in C also come from vertices in C*. That is, there are no edges from outside a source component into such a component.

In the graph of Figure 1 there are three strongly connected components, AB, CD and EF. The component AB is a source component; the components CD and EF are not source components.

Now suppose that C is a source component of the attribute graph of R and that the vertices in C correspond to the attributes in some set $\mathcal{C}$. Clearly, any key of R must functionally determine all the attributes in $\mathcal{C}$. Thus, it makes sense to find all minimal sets of attributes that functionally determine $\mathcal{C}$. Since C is a source component, there is no functional dependency with an attribute in $\mathcal{C}$ on the right side and an attribute not in $\mathcal{C}$ on the left side. Thus, any minimal set of attributes that functionally determine $\mathcal{C}$ cannot contain any attribute that is not in $\mathcal{C}$. Thus the problem of finding minimal sets of keys the functionally determine $\mathcal{C}$ is the problem of finding the keys of a smaller relational database scheme. We call a relational database scheme that can be used to find the minimal sets of keys that functionally determine $\mathcal{C}$ a *restriction* of R to $\mathcal{C}$, *restrict*$(R, \mathcal{C})$. More formally, we have the following:

Definition 6. If R is a relational database scheme with attributes $\mathcal{A}$ and $\mathcal{B} \subseteq \mathcal{A}$, then R' is a restriction of R to $\mathcal{B}$, if $\mathcal{B}$ is the set of attributes of R' and for all $X, Y \subseteq \mathcal{B}$, $(X \rightarrow Y)_{R'}$ if and only if $(X \rightarrow Y)_R$.

If R is the relational database scheme whose attribute graph is in Figure 1, then R' with attributes $\{A, B\}$ and functional dependencies $\{A \rightarrow B, B \rightarrow A\}$ is a restriction of R to $\mathcal{B}$. Restrictions defined this way have the property we want.

Lemma 4. *If $\mathcal{C}$ is the set of attributes corresponding to a source component of the attribute graph of a relational database scheme R and R' is a restriction of R to $\mathcal{C}$, then K is a key of R' if and only if K is a minimal set of attributes that functionally determine $\mathcal{C}$ in R'.*

Proof 4. Let $K \subseteq \mathcal{C}$ be a set of attributes. Note that, by the definition of restriction, $(K \rightarrow \mathcal{C})_{R'}$ if and only if $(K \rightarrow \mathcal{C})_R$. Thus, K is a key or superkey of R' if and only if K functionally determines $\mathcal{C}$ in R.

We also need to show that minimality is preserved. Suppose that K is a minimal set of attributes such that $(K \rightarrow \mathcal{C})_R$, but that K is a superkey, not a key, of R'. But,

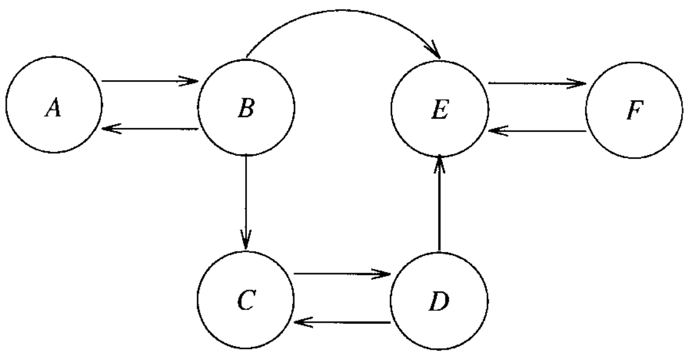

FIGURE 1. An attribute graph.

then there exists $K' \subset K$ that is a key of R'. Moreover, by the definition of restriction, $(K' \to \mathcal{C})_R$, contradicting the minimality of K. Alternatively, suppose that K is a key of R', but $K' \subset K$ satisfies $(K' \to \mathcal{C})_R$. But then, K' is a key or superkey of R', contradicting the fact that K is a key of R'. Therefore, the keys of R' are the minimal sets of attributes that functionally determine $\mathcal{C}$ in R. □

If $\mathcal{C}$ is the set of attributes corresponding to a source component of the attribute graph of R, then a restriction of R to $\mathcal{C}$ is easy to calculate. Let $\mathit{restrict}(R,\mathcal{C})$ be the relational database scheme with attributes $\mathcal{C}$ and whose functional dependencies are calculated by the following procedure. Start with the functional dependencies of R and remove all functional dependencies that contain an attribute in $\bar{\mathcal{C}}$ on the left side. Next remove all attributes in $\bar{\mathcal{C}}$ from any of the functional dependencies that remain. Finally, all functional dependencies with an empty right side are trivial, so remove them. The result is clearly a relational database scheme whose attributes are $\mathcal{C}$, but more is true.

LEMMA 5. *The relational database scheme restrict*$(R,\mathcal{C})$ *defined above is a restriction of R to* $\mathcal{C}$*, provided that* $\mathcal{C}$ *is the set of attributes associated with a source component of* $G[R]$.

Proof 5. Abbreviate $\mathit{restrict}(R,\mathcal{C})$ by R'. Note that every functional dependency of R' is also a functional dependency of R. Thus if $(X \to Y)_{R'}$, then $(X \to Y)_R$.

Proving the converse is more difficult. Suppose that $(X \to Y)_R$. We would like to see that if $X, Y \subseteq \mathcal{C}$, then $(X \to Y)_{R'}$. This functional dependency can be derived from a sequence of functional dependencies $(X \to Z_0)_R$, $(X \to Z_1)_R, (X \to Z_2)_R, \ldots (X \to Z_k)_R$, where $Z_0 = X$, $Y \subseteq Z_k$, and $Z_{i+1} = Z_i \cup V_i$, where $U_i \to V_i$ is a functional dependency of R and $U_i \subseteq Z_i$. We would like to see that $(X \to (Z_0 \cap \mathcal{C}))_{R'}, (X \to (Z_1 \cap \mathcal{C}))_{R'}, (X \to (Z_2 \cap \mathcal{C}))_{R'}, \ldots$ $(X \to (Z_k \cap \mathcal{C}))_{R'}$ is a sequence of functional dependencies in R'. The proof is by induction. Recall that $Z_0 = X \subseteq \mathcal{C}$, so the first functional dependency is just $(X \to X)_{R'}$. Now suppose that $(X \to (Z_i \cap \mathcal{C}))_{R'}$. There are two cases depending on whether $V_i \cap \mathcal{C} = \emptyset$. If $V_i \cap \mathcal{C} = \emptyset$, then $(Z_i \cup V_i) \cap \mathcal{C} = Z_i \cap \mathcal{C}$, so $(X \to (Z_{i+1} \cap \mathcal{C}))_{R'}$, since $Z_{i+1} \cap \mathcal{C} = Z_i \cap \mathcal{C}$. Alternatively, if $V_i \cap \mathcal{C} \neq \emptyset$, then $U_i \subseteq \mathcal{C}$, since $\mathcal{C}$ is the set of attributes from a source component of $G[R]$. Moreover, $(U_i \to (V_i \cap \mathcal{C}))$ is a functional dependency of R'. This means that $U_i \subseteq Z_i \cap \mathcal{C}$, so $(X \to ((Z_i \cap \mathcal{C}) \cup U_i))_{R'}$, so $(X \to ((Z_i \cap \mathcal{C}) \cup (V_i \cap \mathcal{C})))_{R'}$. Thus, in either case, $(X \to (Z_{i+1} \cap \mathcal{C})_{R'})$, and by induction $(X \to (Z_k \cap \mathcal{C}))_{R'}$. Since $Y \subseteq Z_k \cap \mathcal{C}$, we have $(X \to Y)_{R'}$. This is what we needed to show that R' is a restriction of R to $\mathcal{C}$. □

Intuitively, the proof of the preceding Lemma says that since $\mathcal{C}$ is the set of attributes corresponding to a source component of $G[R]$, any derivation of $(X \to Y)_R$ with $Y \subseteq \mathcal{C}$ can be mimicked in R'. This assumption is necessary to the proof of the preceding Lemma. Indeed, the generalization of the Lemma to arbitrary sets of attributes is false.

In the example of Figure 1, recall that $\{A, B\}$ is a source component. Thus $\mathit{restrict}(R, \{A, B\})$ has attributes A and B and the functional dependencies $A \to B$ and $B \to A$.

Once we have found the keys of $\mathit{restrict}(R,\mathcal{C})$, we would like to add attributes to them to make them keys of R. Recall that if X is a set of attributes of R then the *closure* of X, written X^+, is the set of all attributes that X functionally determines. Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be keys of $\mathit{restrict}(R,\mathcal{C})$. Then, $\mathcal{K}_1 \subseteq \mathcal{C}$ and $\mathcal{K}_2 \subseteq \mathcal{C}$. Moreover, $\mathcal{K}_1$ and $\mathcal{K}_2$ each functionally determine $\mathcal{C}$ in R, and, hence, each other. Therefore, $(\mathcal{K}_1^+)_R = (\mathcal{K}_2^+)_R$. In the example of Figure 1, the two keys of $\mathit{restrict}(R, \{A, B\})$ are A and B. It is easy to verify that $A^+ = B^+ = ABCDEF$, so the keys of this relational database scheme are A and B.

This suggests that the problem of extending a key of $\mathit{restrict}(R,\mathcal{C})$ to a key of R, depends only on $\mathcal{C}$ and not on the key to be extended. Intuitively, then the problem of extending a key $\mathcal{K}$ of $\mathit{restrict}(R,\mathcal{C})$ to a key of R corresponds to computation of the keys of another relational database scheme that has attributes in $\bar{\mathcal{C}}$, and whose functional dependencies come from assuming that all attributes in $\mathcal{C}$ are give for free. We call such a relational database scheme a *contraction of R to* $\bar{\mathcal{C}}$. It turns out that not all attributes in $\bar{\mathcal{C}}$ should be in the contraction of R to $\bar{\mathcal{C}}$. Note that if K is a key of $\mathit{restrict}(R,\mathcal{C})$, then there may be an attribute $D \in (K^+ - \mathcal{C})$. If such an attribute were in the contraction of R to $\bar{\mathcal{C}}$, it would be automatically functionally determined by any set of attributes, including the empty set. Since such attributes are not usually in relational database schemes, we omit them from the contraction. More formally, we have:

DEFINITION 7. Let R be a relational database scheme with attributes $\mathcal{A}$ and let $\mathcal{B} \subseteq \mathcal{A}$. A contraction of R to $\bar{\mathcal{B}}$, is a relational database scheme R' whose attributes are $\mathcal{A} - \mathcal{B}^+$ and whose functional dependencies satisfy for all $X, Y \subseteq \mathcal{A} - \mathcal{B}^+$, $(X \to Y)_{R'}$ if and only if $((X \cup \mathcal{B}) \to Y)_R$.

In the first example, $\mathcal{C}^+ = \mathcal{A}$, so the empty relational database scheme with no attributes is a contraction of R to $\bar{\mathcal{C}}$. It is instructive to consider an example with a less trivial contraction. Consider the relational database scheme R_2 with attributes $\{A, B, C, D, E, F, G\}$ and functional dependencies $\{A \to B,\ B \to A,\ B \to D,\ BC \to E,\ DG \to F,\ EF \to G\}$. The attribute graph for this relational database scheme is shown in Figure 2. The attribute graph has two source components, $\{A, B\}$ and $\{C\}$. Suppose that we choose the source component $\{A, B\}$. Then $\mathit{restrict}(R_2, \{A, B\})$ has two keys A and B. In the original relational database scheme R_2, $A^+ = B^+ = \{A, B, D\}$. If we consider a relational database scheme, $\mathit{contract}(R_2, \{C, D, E, F, G\})$ with attributes $\{C, E, F, G\}$ and functional dependencies $\{C \to E, G \to F, EF \to G\}$ it is easy to verify that this relational database scheme is, in fact, a contraction of R_2 to $\{C, D, E, F, G\}$.

Contractions can often be easily calculated. Suppose that we wish to calculate the contraction of a relational database scheme R to $\mathcal{B}$ and $\bar{\mathcal{B}}$ is the set of attributes corresponding to a source component of the attribute graph of R. Consider the

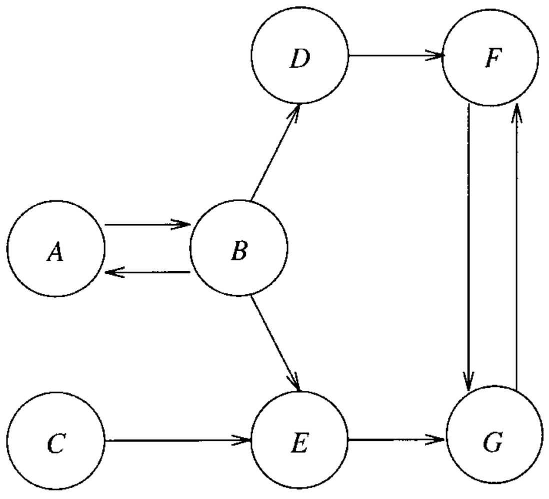

FIGURE 2. Another attribute graph.

relational database scheme $contract(R, \mathcal{B})$ that is calculated by the following steps:

1. Calculate the sets $\bar{\mathcal{B}}$, $(\bar{\mathcal{B}})^+$ and $\mathcal{A}-(\bar{\mathcal{B}})^+$. The set $\mathcal{A}-(\bar{\mathcal{B}})^+$ is the set of attributes of $contract(R, \mathcal{B})$.
2. Remove all attributes not in $\mathcal{A}-(\bar{\mathcal{B}})^+$ from each functional dependency of R.
3. Any resulting functional dependency with an empty right side is trivial, so remove it. The resulting set of functional dependencies is the set of functional dependencies of $contract(R, \mathcal{B})$.

The relational database scheme $contract(R, \mathcal{B})$ calculated this way is a contraction of R to $\mathcal{B}$ as we see in the following:

Lemma 6. *If R is a relational database scheme and $\bar{\mathcal{B}}$ is the set of attributes corresponding to a source component of the attribute graph of R, then $contract(R, \mathcal{B})$ as defined above is a contraction of R to $\mathcal{B}$.*

Proof 6. Let us abbreviate $R' = contract(R, \mathcal{B})$. If $X \rightarrow Y$ is a functional dependency of R and $X \subseteq (\bar{\mathcal{B}})^+$, then $Y \subseteq (\bar{\mathcal{B}})^+$. Thus, if any functional dependency of R attains an empty left side after step 2, it also attains an empty right after that step, so it is removed completely in step 3. Therefore, no functional dependency of R' has an empty right or left side.

Since R' was formed by deleting the attributes in $(\bar{\mathcal{B}})^+$ from the functional dependencies of R, if $(X \rightarrow Y)_{R'}$, then $((X \cup (\bar{\mathcal{B}})^+) \rightarrow Y)_R$. Moreover, $\bar{\mathcal{B}} \rightarrow (\bar{\mathcal{B}})^+)_R$, so if $(X \rightarrow Y)_{R'}$, then $((X \cup \bar{\mathcal{B}}) \rightarrow Y)_R$.

To show that R' is a contraction of R to $\mathcal{B}$ we also need to show the converse. Suppose that $((X \cup \bar{\mathcal{B}}) \rightarrow Y)_R$. Just as in the proof of Lemma 5, the proof goes by taking a derivation that $((X \cup \bar{\mathcal{B}}) \rightarrow Y)_R$ and using it to construct a proof that $(X \rightarrow Y)_{R'}$. Recall that a derivation of a functional dependency, consists of a series of functional dependencies $((X \cup \bar{\mathcal{B}}) \rightarrow Z_0)_R, ((X \cup \bar{\mathcal{B}}) \rightarrow Z_1)_R, ((X \cup \bar{\mathcal{B}}) \rightarrow Z_2)_R$,- $\ldots((X \cup \bar{\mathcal{B}}) \rightarrow Z_k)_R$, where $Z_0 = (X \cup \bar{\mathcal{B}})$, $Y \subseteq Z_k$, and $Z_{i+1} = Z_i \cup V_i$, where $U_i \rightarrow V_i$ is a functional dependency of R and $U_i \subseteq Z_i$. Since $U_i \rightarrow V_i$ in R, we have that $(U_i \cap (\mathcal{A}-(\bar{\mathcal{B}})^+)) \rightarrow (V_i \cap (\mathcal{A}-(\bar{\mathcal{B}})^+))$ in R'. Thus it follows by induction that $(X \rightarrow (Z_0 \cap (\mathcal{A}-(\bar{\mathcal{B}})^+)))_{R'}$, $(X \rightarrow (Z_1 \cap (\mathcal{A}-(\bar{\mathcal{B}})^+)))_{R'}$, $(X \rightarrow (Z_2 \cap (\mathcal{A}-(\bar{\mathcal{B}})^+)))_{R'}, \ldots$ $(X \rightarrow (Z_k \cap (\mathcal{A}-(\bar{\mathcal{B}})^+)))_{R'}$. Since $Y \subseteq Z_k$ and $Y \subseteq (\mathcal{A}-(\bar{\mathcal{B}})^+)$, we have that $(X \rightarrow Y)_{R'}$. This completes the proof that $contract(R, \mathcal{B})$ is a contraction of R to $\mathcal{B}$. □

The previous discussion suggests the following algorithm to compute the keys of a relational database scheme R:

1. Find $G[R]$ the attribute graph of R.
2. If $G[R]$ is strongly connected, then use some other algorithm, e.g. the algorithm of [4]; otherwise do steps 3–7.
3. Let $\mathcal{C}$ be the set of attributes of a source component of $G[R]$.
4. Calculate $restrict(R, \mathcal{C})$ and $contract(R, \bar{\mathcal{C}})$.
5. Calculate $\mathcal{S}_r$, the set of keys of $restrict(R, \mathcal{C})$. The attribute graph of $restrict(R, \mathcal{C})$ will be strongly connected, so use the same algorithm that would have been used for step 2.
6. Recursively, calculate $\mathcal{S}_c$, the set of keys of $contract(R, \mathcal{C})$.
7. Let $\mathcal{S}$ consist all sets of attributes that are the union of a key in $\mathcal{S}_r$ and a key in $\mathcal{S}_c$. More formally, $\mathcal{S}$ is the set of unions $K_r \cup K_s$ for each pair $(K_r, K_s) \in \mathcal{S}_r \times \mathcal{S}_c$. That is, $\mathcal{S} = \{K_r \cup K_c | K_r \in \mathcal{S}_r, K_c \in \mathcal{S}_c\}$. Return $\mathcal{S}$ as the answer.

Clearly, $\mathcal{S}$ will be a set of sets of attributes of R. It is less clear that it contains the correct sets. That is, it contains exactly the keys of R. This means proving that every set in $\mathcal{S}$ is a key of R and that every key of R is in $\mathcal{S}$. The second fact amounts to proving that if K is key of R, then $K_r = K \cap \mathcal{C}$ is a key of $restrict(R, \mathcal{C})$ and $K_c = K \cap \bar{\mathcal{C}}$ is a key of $contract(R, \bar{\mathcal{C}})$. The proof of correctness comes down to proving that certain sets of attributes are keys of certain relational database schemes. It turns out to be easier to first prove that the sets in question are either keys or superkeys and then to prove that they are minimal and, hence, keys.

We can start the proof of correctness by showing that every set in $\mathcal{S}$ is somewhat interesting in that it is a either a key or a superkey.

Lemma 7. *Let $\mathcal{A}$ be the set of all attributes of R and let $\mathcal{C} \subset \mathcal{A}$ be a source component of $G[R]$, the attribute graph of R. Furthermore, let $\mathcal{K}_c$ be a key or superkey of $contract(R, \bar{\mathcal{C}})$ and let $\mathcal{K}_r$ be a key or superkey of $restrict(R, \mathcal{C})$. Then $\mathcal{K}_c \cup \mathcal{K}_r$ is a key or superkey of R.*

Proof 7. Since $\mathcal{K}_c$ is a key or superkey of the contraction $contract(R, \bar{\mathcal{C}})$, the set $\mathcal{K}_c \cup \mathcal{C}$ is a key or superkey of R. Since $\mathcal{K}_r$ is a key or superkey of the restriction $restrict(R, \mathcal{C})$, we have $(\mathcal{K}_r^+)_R \supset \mathcal{C}$. Therefore, $\mathcal{K}_c \cup \mathcal{K}_r$ is a key or superkey of R. □

The previous lemma showed that one can make a key or superkey of R from a key or superkey of $restrict(R, \mathcal{C})$ and a key or superkey of $contract(R, \mathcal{C})$. It is possible to go in the other direction.

LEMMA 8. *Let $\mathcal{A}$ be the set of all attributes of R, let $\mathcal{C} \subset \mathcal{A}$ be a source component of $G[R]$ and let $\mathcal{K}$ be a key or superkey of R. Then $\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+)$ is a key or superkey of contract$(R, \bar{\mathcal{C}})$ and $\mathcal{K} \cap \mathcal{C}$ is a key or superkey of restrict$(R, \mathcal{C})$.*

Proof 8. First, we would like to see that $\mathcal{K} \cap \mathcal{C}$ is a key or superkey of *restrict*$(R, \mathcal{C})$. To save writing complicated subscripts, let $R_r = restrict(R, \mathcal{C})$. Consider a derivation of the fact that $(\mathcal{K}^+)_R = \mathcal{A}$. It consists of a sequence of functional dependencies $(\mathcal{K} \to Z_0)_R, (\mathcal{K} \to Z_1)_R, (\mathcal{K} \to Z_2)_R, \ldots (\mathcal{K} \to Z_k)_R$, where $Z_0 = \mathcal{K}$, $Z_k = \mathcal{A}$, and $Z_{i+1} = Z_i \cup V_i$, where $U_i \to V_i$ is a functional dependency of R and $U_i \subseteq Z_i$. Now we would like to see that $((\mathcal{K} \cap \mathcal{C}) \to (Z_i \cap \mathcal{C}))_{R_r}$. For $i = 0$ this is trivially true. For $i > 0$, we use induction. By induction, $((\mathcal{K} \cap \mathcal{C}) \to (Z_i \cap \mathcal{C}))_{R_r}$. If $V_i \cap \mathcal{C} = \emptyset$, then $(Z_{i+1} \cap \mathcal{C}) = (Z_i \cap \mathcal{C})$, so $((\mathcal{K} \cap \mathcal{C}) \to (Z_{i+1} \cap \mathcal{C}))_{R_r}$ in this case. Alternatively, if V_i contains an attribute in $\mathcal{C}$, then, since $\mathcal{C}$ is the set of attributes corresponding to a source component, $U_i \subseteq \mathcal{C}$. By the definition of restriction, $(U_i \to (V_i \cap \mathcal{C}))_{R_r}$. Thus, in this case, $((\mathcal{K} \cap \mathcal{C}) \to (Z_{i+1} \cap \mathcal{C}))_{R_r}$ too. Therefore, $((\mathcal{K} \cap \mathcal{C}) \to (Z_k \cap \mathcal{C}))_{R_r}$. But, since $Z_k \cap \mathcal{C} = \mathcal{C}$, $\mathcal{K} \cap \mathcal{C}$ is a key of R_r.

Now we would like to see that $\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+)$ is a key or superkey of *contract*$(R, \bar{\mathcal{C}})$. Note that $((\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+)) \cup \mathcal{C} \to (\mathcal{C}^+ \cup (\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+))))_R$. Moreover, $\mathcal{K} \subseteq (\mathcal{C}^+ \cup (\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+)))$. Since $\mathcal{K}$ is a key or superkey of R, $(\mathcal{K} \to \mathcal{A})_R$. Therefore, $((\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+)) \cup \mathcal{C} \to \mathcal{A})_R$, so $\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+)$ is a key of *contract*$(R, \bar{\mathcal{C}})$. □

The correspondence between keys or superkeys of R, *restrict*$(R, \mathcal{C})$ and *contract*$(R, \mathcal{C})$ is close enough that it extends to a correspondence between keys of R, *restrict*$(R, \mathcal{C})$ and *contract*$(R, \mathcal{C})$. Basically, if we delete an attribute from K and the same attribute from either K_r or K_c, we still get keys in both or neither cases. We make this argument precise in the next two lemmas.

LEMMA 9. *Let $\mathcal{A}$ be the set of all attributes of R, let $\mathcal{C} \subset \mathcal{A}$ be a source component of $G[R]$, and let $\mathcal{K}$ be a key of R. Then $\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+)$ is a key of contract$(R, \bar{\mathcal{C}})$ and $\mathcal{K} \cap \mathcal{C}$ is a key of restrict$(R, \mathcal{C})$.*

Proof 9. Because of Lemma 8, we know that $\mathcal{K} \cap \mathcal{C}$ and $\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+)$ are keys or superkeys of *restrict*$(R, \mathcal{C})$ and *contract*$(R, \bar{\mathcal{C}})$, respectively. Now we would like to see that they are minimal.

To see that $\mathcal{K} \cap \mathcal{C}$ is a key of *restrict*$(R, \mathcal{C})$, consider the set $\mathcal{K}'$ consisting of all but one of the attributes in $\mathcal{K} \cap \mathcal{C}$. If $\mathcal{K}'$ is a key of *restrict*$(R, \mathcal{C})$, then by Lemma 7, $\mathcal{K}' = \mathcal{K}' \cup (\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+))$ is a key or superkey of R. Since $\mathcal{K}'$ is a proper subset of $\mathcal{K}$, this contradicts the assumption that $\mathcal{K}$ is a key of R. Therefore, $\mathcal{K} \cap \mathcal{C}$ is minimal and a key of *restrict*$(R, \mathcal{C})$.

Similarly, if we can delete an attribute from $\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+)$ and stillof

$\mathcal{K}$ that is a key of R. Again, this means that $\mathcal{K} \cap (\mathcal{A} - \mathcal{C}^+)$ is a key. □

LEMMA 10. *Let $\mathcal{A}$ be the set of all attributes of R and let $\mathcal{C} \subset A$ be a source component of $G[R]$. Furthermore, let $\mathcal{K}_c$ be a key of contract$(R, \bar{\mathcal{C}})$ and let $\mathcal{K}_r$ be a key of restrict$(R, \mathcal{C})$. Then $\mathcal{K}_c \cup \mathcal{K}_r$ is a key of R.*

Proof 10. This time Lemma 7 shows that $\mathcal{K}_c \cup \mathcal{K}_r$ is a key or superkey of R, and again we need to see that it is minimal. If we delete an attribute from $\mathcal{K}_c$ to form $\mathcal{K}_c'$ and $\mathcal{K}_c' \cup \mathcal{K}_r$ is still a key of R, then by Lemma 8, $\mathcal{K}_c'$ is a key or superkey of *contract*$(R, \bar{\mathcal{C}})$. Similarly, if we delete an attribute from $\mathcal{K}_r$ to form $\mathcal{K}_r'$ and $\mathcal{K}_c \cup \mathcal{K}_r'$ is still a key of R, then by Lemma 8, $\mathcal{K}_r'$ is a key or superkey of *restrict*$(R, \mathcal{C})$. In either case we have a contradiction, so $\mathcal{K}_c \cup \mathcal{K}_r$ must be minimal and must be a key of R. □

A corollary of the preceding four lemmas is the following theorem that characterizes the keys of a relational database scheme.

THEOREM 1. *Let $\mathcal{C}$ be a source component of $G[R]$. Then all keys of R are the union of a key of restrict$(R, \mathcal{C})$ and a key of a contract$(R, \mathcal{C})$. Moreover, all such unions are keys of R.* □

Thus to compute the keys of R it suffices to: compute $G[R]$, find the strongly connected components of $G[R]$, find a source component $\mathcal{C}$, and compute the keys of *contract*$(R, \bar{\mathcal{C}})$ and *restrict*$(R, \mathcal{C})$.

The attribute graph of *restrict*$(R, \mathcal{C})$ will always be strongly connected. Thus another algorithm is needed to compute the keys of this database scheme. One obvious candidate is the algorithm given in [4]. Suppose that we use this algorithm to find the keys for both *restrict*$(R, \mathcal{C})$ and *contract*$(R, \bar{\mathcal{C}})$. The running time will then be $O(F_1K_1A_1(K_1 + A_1) + F_2K_2A_2(K_2 + A_2) + K_1K_2A + FA)$, where F_1, A_1 and K_1 are the number of functional dependencies, attributes and keys of *restrict*$(R, \mathcal{C})$, and F_2, A_2 and K_2 are the number of functional dependencies, attributes and keys of *contract*$(R, \bar{\mathcal{C}})$. The first two terms come from the time required by the calls to the Lucchesi and Osborn algorithm, the third term is just the time required to write out the answer, and the fourth term is the time required to compute the strongly connected components of $G[R]$ and to compute the restriction *restrict*$(R, \mathcal{C})$ and the contraction *contract*$(R, \bar{\mathcal{C}})$. Note that an attribute is in exactly one of $\mathcal{C}$ and $\bar{\mathcal{C}}$, so $A_1 + A_2 \le A$. Similarly, a functional dependency of R leads to a functional dependency of *restrict*$(R, \mathcal{C})$ or a functional dependency of *contract*$(R, \bar{\mathcal{C}})$, but not both. Thus $F_1 + F_2 \le F$. Finally, a consequence of Theorem 1 is that $K_1K_2 = K$. This means that if $G[R]$ is not strongly connected, then using Theorem 1 is guaranteed to save time.

It is often the case that the attribute graph of *contract*$(R, \bar{\mathcal{C}})$ is not strongly connected. In this case, it is better to use the algorithm presented here recursively to calculate the keys of *contract*$(R, \bar{\mathcal{C}})$.

For some applications (e.g. finding the key with the fewest attributes) it is not necessary to list all the keys explicitly. In this case, step 7 can be omitted. Even though the running time for this step is proportional to the length of the output, it can be the most time- consuming step. If the algorithm is used recursively to calculate the keys of the contraction, the result will be a set of sets of keys of disjoint pieces of the original relational database scheme. A key of the original relational database scheme consists of the union of one key from each of these sets. If each of these sets of keys of parts of the relational database scheme contains only a polynomial number of keys, then each call to the Lucchesi and Osborn algorithm will run in polynomial time and the whole algorithm to find the keys of the original relational database scheme will run in polynomial time. Note that there are 2^k subsets of k keys, so a relational database scheme with k attributes has fewer than 2^k keys. Thus a sufficient condition for the algorithm presented here to run in polynomial time is for each strongly connected component of the attribute to have constant size. In this case, the number strongly connected components will be linear in the number of attributes of the relational database scheme. If a constant fraction of the calls to the Lucchesi and Osborn algorithm produce two or more keys, the whole relational database scheme will have an exponential number of keys, but the algorithm can find a representation of them in polynomial time.

The next section presents an example of how these theorems can be used. We use the above algorithm to find the candidate keys of the given relation scheme. For this example, a universal relation scheme R with a set of FDs Γ are given. The example is chosen such that $\mathcal{B}$ would contain all attributes of R.

EXAMPLE 3

Given schema $R(ABCDEFG)$ and $\Gamma = \{A \to B, AB \to C, BC \to A, AC \to D, ED \to F, EF \to G, AG \to E\}$, find all candidate keys of R.

Solution

The attribute graph for this scheme is shown in Figure 3. Note that it has three strongly connected components, ABC, D and EFG. The only source component is ABC. A restriction of R to ABC has $\Gamma = \{A \to B, AB \to C, BC \to A\}$. Experimenting, we see that A is a key of $restrict(R, ABC)$, but that B and C are not keys. Furthermore, BC is a key of $restrict(R, ABC)$. Now we need to compute a contraction $contract(R, DEFG)$. Note that $(ABC)^+ = ABCD$, so the contraction $contract(R, DEFG)$ has the dependencies $\{E \to F,\ EF \to G, G \to E\}$. Direct calculation shows that the keys of $contract(R, DEFG)$ are E and G. This means that the whole schema has four keys AE, AG, BCE and BCG.

5. CONCLUSIONS

We showed that a simple categorization of attributes into the sets $\mathcal{L}, \mathcal{R}$ and $\mathcal{B}$ allows us to distinguish between those attributes that will participate in the candidate keys of a relational database schema and those that do not. This categorization will provide a simple and efficient solution for finding the keys when the sets $\mathcal{L}$ and $\mathcal{R}$ are not empty.

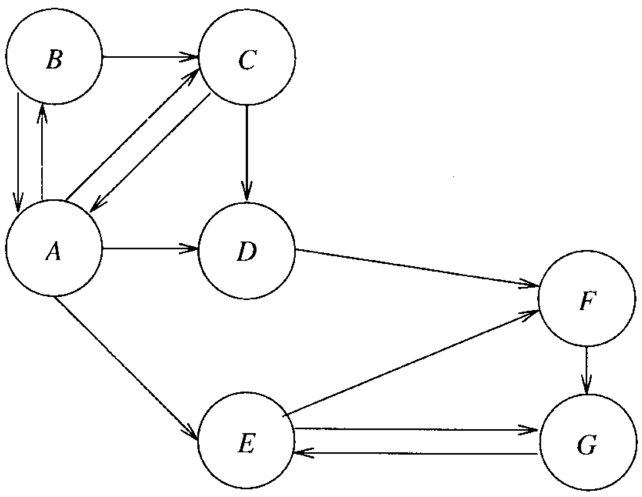

FIGURE 3. The attribute graph of R.

When these two sets are empty, i.e. $\mathcal{B}$ contains all attributes, we face the worst-case situation.

To handle the worst-case situation we introduced a graph algorithm to show that it is possible to simplify the problem of finding all possible keys of a database schema, if the attribute graph of the functional dependencies is not strongly connected.

We plan to work on a more cost-effective algorithm than a simple brute force search in the case that the attribute graph is strongly connected.

ACKNOWLEDGEMENTS

Comments by anonymous reviewers have substantially improved the quality of this article; we sincerely appreciate their concerns and assistance. Hossein Saiedian was partially supported by the UCR under grant #9330-1995, University of Nebraska at Omaha. Thomas Spencer was supported by the NSF under grant CCR-9319772 and by the University Committee on Research, University of Nebraska at Omaha.

REFERENCES

[1] Armstrong, W. (1974) Dependency structures of database relationships. In *Proc. 1974 IFIP Congress*, pp. 580–583.

[2] Ullman, J. (1988) *Principles of Database and Knowledge-Based Systems*, volume I. Computer Science Press, Rockville, MD.

[3] Maier, D. (1983) *The Theory of Relational Databases*. Computer Science Press, Rockville, MD.

[4] Lucchesi, C. and Osborn, S. (1978) Candidate keys for relations. *Journal of Computer and Systems Sciences*, **17**(2), 270–279.

[5] Elmasri, R. and Navathe, S. B. (1994) *Fundamentals of Database Systems*. Benjamin-Cummings, Menlo Park, CA, 2nd edition.

[6] Kundu, S. (1985) An improved algorithm for finding a key of a relation. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 189–192. ACM.

[7] Ausiello, G., D'atri, A. and Saccà, D. (1983) Graph algorithms for functional dependency manipulation. *Journal of the ACM*, **30**(4), 752–766.

[8] Biskup, J., Demetrovics, J., Libkin, L. and Muchnik, I. (1991) On relational database schemes having unique minimal key. *J. Information Processing Cybernetics*, **27**, 217–225.

[9] Maier, D. (1980) Minimum covers in the relational database model. *Journal of the ACM*, **27**(4), 664–674.