Design by Example: An Application of Armstrong Relations

Heikki Mannila
Kari-Jouko Räihä
TR 85-665
March 1985

Department of Computer Science
Cornell University
Ithaca, New York 14853

* This work was partially supported by the Academy of Finland. An extended abstract of this paper was presented at the Fourth ACM SIGACT-SIGMOD Conference on Principles of Database Systems, Portland, Oregon, March 1985.

1 Department of Computer Science, University of Helsinki, Tukholmankatu 2, SF-00250 Helsinki 25, Finland.
2 On leave from the Department of Computer Science, University of Helsinki.
DESIGN BY EXAMPLE:

AN APPLICATION OF ARMSTRONG RELATIONS

by

Heikki Mannila
Department of Computer Science, University of Helsinki
Tukholmankatu 2, SF-00250 Helsinki 25, Finland

and

Kari-Jouko Räihä
Department of Computer Science, Cornell University
405 Upson Hall, Ithaca, N.Y. 14853

Abstract. Example relations, and especially Armstrong relations, can be used as user-friendly representations of dependency sets. In this paper we analyze the use of Armstrong relations in database design with functional dependencies, and show how they and the usual representation of dependencies can be used together. Special attention is given to the size of Armstrong relations. We derive new bounds for the size of minimal Armstrong relations for normalized schemes. New algorithms are also given for generating Armstrong relations and for inferring the functional dependencies holding in a relation.

Categories and Subject Descriptors: H.2.1 [Database Management]: Logical Design - normal forms, schema and subschema

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Armstrong relation, functional dependency, relational database, computational complexity

---

1 This work was partially supported by the Academy of Finland. An extended abstract of this paper was presented at the Fourth ACM SIGACT-SIGMOD Conference on Principles of Database Systems, Portland, Oregon, March 1985.

2 On leave from the Department of Computer Science, University of Helsinki.
1. Introduction

A broad class of dependency types has been developed for the relational data model (see e.g. [Ul82]). In general, a dependency is said to hold for a relation scheme $R$ if and only if it holds for all corresponding relations. Consequently, an arbitrary relation can only show that a particular dependency does not hold for the scheme, but not the opposite.

Armstrong relations [Ar74] are special relations that can show both the nonexistence and the existence of dependencies for a relation scheme. That is, a dependency holds for the relation scheme if and only if it holds for the Armstrong relation. The existence of Armstrong relations for sets of functional dependencies was shown in [Ar74]. Fagin [Fa82b] has shown the existence of Armstrong relations for a wide class of dependency types; however, there exist some dependency types that do not always allow Armstrong relations. In this paper we will restrict ourselves to sets of functional dependencies and their Armstrong relations.

By their definition, Armstrong relations can be regarded as a representation of the closure of a dependency set. Since this representation is in the form of an example relation, it should be useful for the database designer. In Section 2 we analyze the different ways of using Armstrong relations in the design process. Even in an almost trivial example the benefits of using example relations in general and Armstrong relations in particular become obvious. Our study also gives rise to some interesting new research problems.

Armstrong relations have a drawback when compared to the conventional linear representation of a dependency set. In [BDFS84] it is shown that for some dependency sets, the size of all Armstrong relations must be exponential (both in the number of attributes and in the number of dependencies). However, from the practical point of view the example dependency set used for establishing the lower bound in [BDFS84] can be regarded as irrelevant: it represents a highly unnormalized relation scheme. We review some of the results of [BDFS84] in Section 3. We will also provide a new characterization that turns out to be particularly useful for the algorithms that follow.
Based on the results in Section 3 and motivated by the discussion in Section 2, we derive new bounds for the size of Armstrong relations for normalized relation schemes in Section 4. The results show that the size of a minimal Armstrong relation for a normalized scheme $R$ depends strongly on the number of keys for $R$. In particular, the possibly exponential size of a minimal Armstrong relation depends only on the number of dependencies, not on the number of attributes. This is an encouraging result, since the number of attributes is typically much bigger than the number of dependencies.

Because of the general exponentiality result [BDFS84], we cannot design an algorithm for generating Armstrong relations in polynomial time in the worst case. However, most existing algorithms (e.g. [BDFS84]) require exponential time even in the best case. This is unfortunate, since we have just argued that in reality Armstrong relations can be small. It turns out that there is a fairly simple algorithm for generating Armstrong relations that is much more conservative in its use of time than the existing algorithms. Moreover, the size of the generated Armstrong relation is at most the square of the size of a minimal Armstrong relation. Thus, if a dependency set has a small Armstrong relation, our algorithm is capable of generating a (reasonably) small Armstrong relation faster than previous algorithms. The algorithm is given and analyzed in Section 5.

One new problem discovered by our study in Section 2, dependency inference, is discussed in Section 6. We conclude in Section 7 by summarizing the results and by pointing out various ways for extending the work reported here.

For a survey of Armstrong relations, the reader is referred to [Fa82a]. We assume that the reader is familiar with the basic concepts of relational databases (see e.g. [Ul82] or [Ma83]).

2. The role of Armstrong relations in database design

In the traditional paradigm for designing a relational database scheme, the database designer determines a set of attributes and a set of data dependencies among the attributes. This information can be given as input to an algorithm that produces a database scheme satisfying a chosen
set of properties, e.g. losslessness of join, preservation of dependencies, acyclicity, one of many different normal forms, etc.

In this paper we are not concerned with any particular set of properties, but with the approach as a whole. It has been criticized on various grounds, e.g. because of the difficulty of finding the proper attributes and dependencies, and because often the desired set of properties can be achieved by various designs, some of which are in practice more desirable than others. Clearly, no algorithm can be applied as an ultimate solution to the design problem: the designer must be given the opportunity to express his opinion about the suggested design, and to demand changes if the solution is not satisfactory.

One undeniable virtue of this approach is that it automates a considerable part of the design process. It is possible to build a design tool that helps the designer in his task. Our thesis is that the usefulness of such a tool depends crucially on the form that is used for representing the available information about the database scheme. With a proper representation the above mentioned problems become less severe.

As a running example, let us consider the design of a database scheme for storing information about COURSEs, lecture HOURs, lecture ROOMs, and TEACHERs. Suppose that besides the attributes, the database designer has been able to find the functional dependencies HOUR ROOM $\rightarrow$ COURSE (only one course can be taught in any classroom at any given time) and TEACHER $\rightarrow$ COURSE (each teacher only teaches one course).

Let us suppose that this information is given as input to an algorithm that decomposes the universal relation scheme losslessly into relation schemes in Boyce-Codd normal form. If the second dependency is used as the basis of decomposition, the result would consist of two relation schemes: $\{\text{TEACHER COURSE}\}$ and $\{\text{HOUR ROOM TEACHER}\}$. A relation stored according to the first scheme should satisfy the dependency TEACHER $\rightarrow$ COURSE, whereas nothing is required from the second relation.

This is the information that a design algorithm is typically expected to produce. Suppose it is given to the designer; what kind of a reaction could be expected? The designer has already
expressed the data dependencies to the best of his ability: therefore it is unlikely that he will find any logical anomalies in the proposed schemes. However, the design contains the scheme \{TEACHER \ HOUR \ ROOM\} with no enforced dependencies. This is likely to cause problems.

Our fundamental suggestion is extremely simple: in addition to the relation schemes and their dependencies, the design tool should show the designer examples of relations that can be stored according to the schemes. We shall see that such examples can easily help the designer in detecting potential anomalies in the candidate scheme.

The next obvious question to ask is what kind of example relations should be used. Choosing an arbitrary relation is not going to be very helpful; on the contrary, it can be downright dangerous. For suppose that we show the designer the following example relation filled with arbitrary tuples.

<table>
<thead>
<tr>
<th>TEACHER</th>
<th>HOUR</th>
<th>ROOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ullman</td>
<td>Tu 10:10</td>
<td>Ives 120</td>
</tr>
<tr>
<td>Fagin</td>
<td>We 11:25</td>
<td>Ives 120</td>
</tr>
</tbody>
</table>

Since no data dependencies were required to hold in the relation, this is an example of a legal relation. However, it should not be shown to the designer, since it gives the false impression that the suggested design is acceptable. Even though no dependencies were required to hold in the relation, this particular relation satisfies quite a few nontrivial dependencies: \(\text{HOUR} \rightarrow \text{ROOM} \ \text{TEACHER}, \ \text{TEACHER} \rightarrow \text{HOUR} \ \text{ROOM}\), and all the dependencies derivable from these.

A good example relation should not leave the designer any illusions about what can be stored in the database. Therefore the example relation should satisfy exactly the dependencies that can be derived from the given set of dependencies: no more, no less. In other words, the example relation should be an Armstrong relation.

An Armstrong relation for the empty set of dependencies in the relation scheme \{TEACHER \ HOUR \ ROOM\} is given below.
<table>
<thead>
<tr>
<th>TEACHER</th>
<th>HOUR</th>
<th>ROOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ullman</td>
<td>Tu 10:10</td>
<td>Ives 120</td>
</tr>
<tr>
<td>Fagin</td>
<td>Tu 10:10</td>
<td>Ives 120</td>
</tr>
<tr>
<td>Fagin</td>
<td>We 11:25</td>
<td>Ives 120</td>
</tr>
<tr>
<td>Ullman</td>
<td>Tu 10:10</td>
<td>Hollister 110</td>
</tr>
</tbody>
</table>

From this relation it is easy to see that everything is not as it should be: Ullman is supposed to be in two places at the same time. Since most of us have not yet learnt how to do that, the design should probably be changed. We will return to this issue in a moment, but let us first emphasize some points that relate to the above scenario.

Historically, the first proposal for using Armstrong relations in the design process was made by Silva and Melkanoff [SM81]. They suggest that the design tool should produce Armstrong relations for all the relation schemes, and then join the relations into a universal relation. Showing this relation to the designer should help him in detecting potential anomalies in the database scheme.

However, note that the relations are intended for the database designer, not for the end user. Even though the user interface may be a universal relation, the designer must certainly be aware of the actual relation schemes. It is therefore more useful to produce Armstrong relations for all the relation schemes in the database scheme, instead of a universal relation. Since one property that we usually require from the relation schemes is that they are in normal form, the problem of generating Armstrong relations for normalized relation schemes becomes particularly interesting. The properties of such Armstrong relations are studied in Section 4.

Noble [No83] is another paper emphasizing the value of example relations. The usefulness of illustrative examples for understanding and designing algorithms in general is discussed in [BS85].

A possible problem with our proposal is that some of the original dependencies may not be embodied in the database relations. For example, it is not always possible to obtain a dependency preserving, lossless decomposition of a universal scheme into BCNF schemes. The problem can be overcome by listing explicitly the dependencies that are not preserved. It should be emphasized that we do not propose to replace dependency sets by corresponding Armstrong relations. Since
they are just two representations of the same thing, both should be shown to the designer, who can then base his reasoning on whichever representation he prefers.

For a given set of dependencies there exist many Armstrong relations. Which one of these is most useful? Clearly, the relation should be small if the designer is going to draw any conclusions from it. What, then, is the proper measure of smallness? The number of tuples in the relation is an obvious candidate, which we shall adopt here, too. An alternative might be the number of different values appearing in the relation. The fewer values are used, the more intricate connections between the attributes might be spotted by the designer. To our knowledge, the trade-off between minimizing the number of tuples and the number of values has not been studied at all. Exponential lower bounds for both quantities are proved in [BDFS84].

Even if we aim at Armstrong relations with few tuples, it is not clear that the minimal relation is the most useful for the designer. For instance, it might be illustrative to show the nonexistence of a functional dependency $X \rightarrow Y$ by a pair of consecutive tuples that agree on $X$ and disagree on $Y$. In our example the anomaly would be even easier to notice if the first and fourth tuples were adjacent.

In a minimal relation one tuple may be used as a pair of many other tuples for showing that dependencies do not hold. Therefore it is impossible to juxtapose all the tuples that form pairs. Consequently, it becomes more difficult for the designer to locate the anomalies. Practical experience is needed to gain some guidelines for a suitable structure of an Armstrong relation. Although the algorithms described in the sequel are currently being implemented in a database design tool, we lack such empirical information at the moment. Therefore we will concentrate on the generation of small Armstrong relations without caring about their structure.

Let us now turn to the next step in the design process. Returning to our example, the reason for the anomalous relation is that the designer had forgotten the dependency $TEACHER\ \ HOUR \rightarrow ROOM$ from his set of dependencies. The conventional solution would be to modify the dependency set and run the algorithm again. However, since the anomaly is found by inspecting the Armstrong relation, it appears more natural to remove the anomaly by modify-
ing the relation. That is, the user may replace a relation by another that does not contain anomalous tuples.

This possibility gives rise to some interesting new problems. Suppose first that the designer decides to remove the anomaly by deleting the fourth tuple. The result is acceptable; therefore the designer asks the design tool to treat the modified relation as the Armstrong relation. Thus we have a problem that is opposite to the normal generation problem: given a relation \( r \), the system should find a (small) set of dependencies \( F \) such that \( r \) is an Armstrong relation for \( F \). Note that this problem always has a solution. An algorithm that solves this *dependency inference* problem is necessary if we wish to directly exploit the information in the relations without asking the designer to modify the dependencies for a rerun of the algorithm. Such an algorithm is discussed in Section 6.

In our example, the removal of the fourth tuple leaves a relation that happens to be an Armstrong relation for the dependency set \{\textit{TEACHER} \rightarrow \textit{ROOM}, \textit{HOUR} \rightarrow \textit{ROOM}\}. The "correction" has been too gross: it has introduced an incorrect dependency set. Instead of redesigning the database scheme using this dependency set, it is probably better that the design tool gives the designer this list of dependencies and asks whether that is really what the designer wanted. This is another example of the dual nature of Armstrong relations and dependency sets: some things are easier to see from one, and some things from the other. A good design tool should not attempt to enforce either representation, but provide them both.

When the designer is presented with the above dependencies, he should take some corrective action. To avoid the two incorrect dependencies, the relation should have a tuple that contains a \textit{TEACHER} value and an \textit{HOUR} value that already exist in the relation, combined with a new \textit{ROOM} value. Unlike in our second relation, the \textit{TEACHER} and \textit{HOUR} values should come from different tuples, not from the same tuple. This would yield a relation of the following form:
<table>
<thead>
<tr>
<th>TEACHER</th>
<th>HOUR</th>
<th>ROOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ullman</td>
<td>Tu 10:10</td>
<td>Ives 120</td>
</tr>
<tr>
<td>Fagin</td>
<td>Tu 10:10</td>
<td>Ives 120</td>
</tr>
<tr>
<td>Fagin</td>
<td>We 11:25</td>
<td>Ives 120</td>
</tr>
<tr>
<td>Ullman</td>
<td>We 11:25</td>
<td>Hollister 110</td>
</tr>
</tbody>
</table>

This is an Armstrong relation for $\text{TEACHER \ HOUR} \rightarrow \text{ROOM}$. If additional dependencies, e.g. $\text{HOUR \ ROOM} \rightarrow \text{TEACHER}$, were desirable, the process could be continued. However, we stop at this point.

Even if the modified relation is an Armstrong relation for the desired set, it may still contain unnecessary tuples that serve no purpose (except that they prevent the designer from seeing the true effect of the modifications). It would therefore be useful to have an algorithm for minimizing a given Armstrong relation. The result $s$ of minimizing $r$ should be structurally similar with $r$; ideally, we should have $s \subseteq r$. We shall briefly discuss a simple and efficient minimization algorithm at the end of Section 5.

Before turning to the properties of normalized schemes and to the algorithms, let us close this section by a discussion of one more qualitative property that a good example relation should enjoy. For that purpose, recall that a relation $r$ satisfies a functional dependency $X \rightarrow Y$ if for arbitrary tuples $t_1$ and $t_2$ in $r$, $t_1[X] = t_2[X]$ implies $t_1[Y] = t_2[Y]$. There are two ways for a relation $r$ to satisfy this condition. First, if there exist distinct tuples $t_1$ and $t_2$ in $r$ such that $t_1[X] = t_2[X]$, and if $t_1[Y] = t_2[Y]$ for all such $t_1, t_2$, we say that $r$ satisfies $X \rightarrow Y$ actively. Otherwise, i.e. if $t_1[X] \neq t_2[X]$ for all distinct $t_1, t_2$, the relation $r$ is said to satisfy $X \rightarrow Y$ vacuously. (The terms are adapted from [Ma83]).

It would be useful if the example relations showed the existence and nonexistence of dependencies as explicitly as possible. Consider our final example relation above: since none of the tuples agree on the attribute set $\{\text{TEACHER, HOUR}\}$, the relation satisfies the dependency $\text{TEACHER \ HOUR} \rightarrow \text{ROOM}$ vacuously. Fortunately, this less than desirable behavior is caused by the special nature of the dependency: $\{\text{TEACHER, HOUR}\}$ is a key of the relation scheme. No relation can satisfy actively any functional dependency whose left-hand side contains a key, since the tuples required for active satisfaction are identical and only one of them can be stored.
For the remaining dependencies things are brighter. Let \( X \rightarrow Y \in F \) be such that \( X \) does not contain a key of \( R \), and let \( r \) be an Armstrong relation for \( R \) and \( F \). It follows that for some \( A \) in \( R - X, X \not\rightarrow A \). Because \( r \) is an Armstrong relation, it must contain two tuples, \( t_1 \) and \( t_2 \), which agree on \( X \) and disagree on \( A \). On the other hand, since \( r \) satisfies \( F \), \( t_1 \) and \( t_2 \) must agree on \( Y \), too. Thus \( X \rightarrow Y \) is actively satisfied.

To summarize the preceding discussion, all Armstrong relations do an equally good job of showing the satisfaction of dependencies, up to the ordering of the tuples. For our small example the use of an Armstrong relation helped in detecting an incomplete dependency set. It did not affect the set of relation schemes produced, but for larger and more realistic attribute sets this can easily happen.

3. Characterisations of Armstrong relations

Most of the existing results about the size and generation of Armstrong relations are based on an extremely useful characterization from [BDFS84]. We will first briefly review the characterization, and then restate part of it in a form that can be used to provide new size bounds and generation algorithms. We start by giving some definitions.

Let \( R \) be a relation scheme and \( F \) a set of functional dependencies over \( R \). Then a set \( V \subseteq R \) is \( F \)-closed, if for all \( A \in R \) we have \( V \rightarrow F A \) if and only if \( A \in V \). The set of all \( F \)-closed subsets of \( R \) is denoted by \( CL(F) \); it is closed under intersection. Thus there is a unique minimal subfamily of generators \( GEN(F) \subseteq CL(F) \) such that each member of \( CL(F) \) can be expressed as an intersection of sets in \( GEN(F) \). Note that \( X \in CL(F) \) is in \( GEN(F) \) if and only if \( X \) is properly contained in the intersection of all sets \( Y \in CL(F) \) properly containing \( X \).

As an example, for \( F = \{ \text{TEACHER HOUR} \rightarrow \text{ROOM} \} \) we have

\[
CL(F) = \{ \emptyset, \{ \text{TEACHER} \}, \{ \text{HOUR} \}, \{ \text{ROOM} \}, \{ \text{TEACHER, ROOM} \}, \{ \text{HOUR, ROOM} \} \},
\]

\[
GEN(F) = \{ \{ \text{TEACHER} \}, \{ \text{HOUR} \}, \{ \text{TEACHER, ROOM} \}, \{ \text{HOUR, ROOM} \} \}.
\]

Note that \( \{ \text{TEACHER, HOUR, ROOM} \} \) is considered to be the intersection of an empty
collection of sets. Therefore it is in $CL(F)$ but not in $GEN(F)$.

Let $r$ be a relation over $R$. Then

$$agr(r) = \{ X \mid \text{for distinct } t_1, t_2 \text{ in } r, t_1 \text{ and } t_2 \text{ agree exactly on } X \}.$$  

For example, in the last example relation $r$ in Section 2 we have $agr(r) = \emptyset, \{TEACHER\}, \{HOUR\}, \{ROOM\}, \{TEACHER, ROOM\}, \{HOUR, ROOM\}$.

**Lemma 1.** [BDFS84] Let $r$ be a relation over $R$. Then $r$ is an Armstrong relation for $F$ if and only if $GEN(F) \subseteq agr(r) \subseteq CL(F)$. \qed

As it stands, the definition of $GEN(F)$ is highly nonconstructive. We will next give a theorem showing that this family can be defined more operationally. Let $F$ be a set of FDs over $R$ and $A$ an attribute of $R$. Define

$$max(F,A) = \{ Y \subseteq R \mid Y \text{ is a nonempty maximal set (with respect to } \subseteq)$$

$$\text{such that } Y \not\rightarrow^F A \}.$$  

Let $MAX(F)$ be the union of the sets $max(F,A)$, where $A \in R$. We have the following result.

**Theorem 1.** $MAX(F) = GEN(F)$ for all $F$.

**Proof.** Let $Y \in max(F,A)$ for some $A$. First, $Y$ is $F$-closed. If it were not, for some $B \notin Y$ we would have $Y \rightarrow B$. Then $YB \not\rightarrow A$, since otherwise $Y \rightarrow A$. But $Y \subseteq YB$, contradicting the maximality of $Y$. Secondly, if $W$ properly contains $Y$ and $W$ is $F$-closed, then $A \in W$ again by maximality of $Y$. So $Y$ is properly contained in the intersection of all $F$-closed sets that properly contain $Y$, and thus $Y$ is a member of $GEN(F)$.

Conversely, let $X$ be an arbitrary member of $GEN(F)$. Let $R - X = \{ A_1, \ldots, A_k \}$. As $X$ is closed, $X \not\rightarrow A_i$ for all $i$. Let $Y_i$ be for each $i$ a member of $max(F,A_i)$ extending $X$. By the previous paragraph each $Y_i$ belongs to $GEN(F)$. Let $Y$ be the intersection of the $Y_i$s; it includes $X$. Actually $Y = X$, since $A_i \notin Y$, for all $i$. Thus $X$ is expressible as an intersection of sets in $GEN(F)$. As $X$ itself is in $GEN(F)$, this implies that $X$ is $Y_i$ for some $i$. But the $Y_i$s are in $MAX(F)$, so the equality $MAX(F) = GEN(F)$ has been proved. \qed
4. Size bounds for normalized Armstrong relations

Lemma 1 yields bounds for the size of a minimal Armstrong relation. Using an argument based on the inclusion $GEN(F) \subseteq \text{agr}(r)$, it is shown in [BDFS84] that the number of tuples in a minimal Armstrong relation is at most $|GEN(F)| + 1$ and at least $\sqrt{1 + (1 + 8|GEN(F)|)}$. An example is also given in [BDFS84] showing that $|GEN(F)|$ (and thus the size of a minimal Armstrong relation) can indeed be exponential, both in the number of attributes in the scheme, and in $|F|$. In this section we will show that for a normalized relation scheme the size of $GEN(F)$ depends on the number of keys and is thus usually small.

Recall that a relation scheme $R$ is in BCNF [Co74] if for each dependency $X \rightarrow Y$ holding in $R$, the left-hand side $X$ contains a key.

**Theorem 2.** Let $R$ be in BCNF with respect to $F$, let $X_1, \cdots, X_k$ be the keys of $R$, and let $Y$ be the set of nonprime attributes, i.e. $Y = R - X_1 - \cdots - X_k$. The size of $GEN(F)$ is at most

$$\left(\prod_{i=1}^{k} |X_i|\right)(|Y| + 1).$$

**Proof.** A set $W \subseteq R$ is closed if and only if it does not contain any key. So the sets in the following family are all closed:

$$P = \{R - \{A_1, \cdots, A_k\} | A_i \in X_i \text{ for all } i\} \cup \{R - \{A_1, \cdots, A_k, B\} | A_i \in X_i \text{ for all } i \text{ and } B \in Y\}.$$ 

Clearly, $|P| = \left(\prod_{i=1}^{k} |X_i|\right)(|Y| + 1)$; we will prove the theorem by showing that $P = GEN(F)$.

If $W \in P$ and $Y \subseteq W$, then $W \not\vdash C$ for any $C \notin W$ and, moreover, $W$ is a maximal set with this property. If $W \in P$ and for some $B \in Y$ we have $B \notin W$, then $W \not\vdash B$ and again $W$ is a maximal set with this property. So all elements of $P$ belong to the family $MAX(F)$, which by Theorem 1 equals $GEN(F)$. Conversely, an element of $MAX(F)$ is easily seen to belong to $P$, thus proving the theorem. $\square$

For BCNF relation schemes it is quite reasonable to assume that there are few keys. For example, a BCNF scheme can have only as many keys as there are dependencies, while an arbitrary relation scheme can have an exponential (compared to the number of dependencies) number
of keys. In fact, Theorem 2 shows that the size of a minimal Armstrong relation for a BCNF scheme $R$ is exponential only in $|F|$, not in $|R|$. As an example, consider the scheme with the dependency $TEACHER \rightarrow ROOM$ from Section 2. It is in BCNF, and our expression gives the bound 2·2=4 for the $GEN(F)$ family. In the beginning of Section 3 we saw that the $GEN(F)$ family does indeed have four elements.

If we assume that $R$ is in third normal form [Co72], we get a similar upper bound for the size of $GEN(F)$.

**Theorem 3.** Let $R$ be in 3NF with respect to $F$, and let $X_i$ and $Y$ be as in Theorem 2. Denote by $Z_i$, $i=1, \cdots, m$ the sets that are not keys but still imply some (prime) attribute. Then the size of $GEN(F)$ is at most

$$\frac{1}{3} \prod_{i=1}^{k} |X_i| \cdot \prod_{j=1}^{m} (|Z_j|+1)(|Y|+1).$$

\[\square\]

5. Generating Armstrong relations

Several algorithms have appeared for generating Armstrong relations, but they all suffer from efficiency problems. For example, Grant and Jacobs [GJ82] start from a relation that has a pair of tuples for each dependency that does not belong to $F^*$; here $F^*$ denotes the closure of $F$. Each pair implies that the corresponding dependency does not hold in the relation. The relation is then chased using $F$, and the result is an Armstrong relation for $F$. Obviously, the size of the initial relation is easily exponential.

The generation algorithm given in [BDFS84] works in two steps: it first computes $CL(F)$, and then constructs a relation $r$ such that $agr(r) = CL(F)$. By Lemma 1, $r$ is the desired Armstrong relation for $F$.

For the first part the algorithm in [BDFS84] uses a brute force method: it considers all the subsets of $R$ and checks which of them are closed with respect to $F$. Thus the time complexity of the algorithm is exponential even in the best case. The authors note that they "could get away
with using $\text{GEN}(F)$ instead of $\text{CL}(F)$ in the construction, but [they] do not wish to spend the time to prune out the nongenerators". We wish to carry this strive for efficiency a step further, and avoid computing $\text{CL}(F)$ by computing $\text{GEN}(F)$ directly. This is possible because of the new characterization given in Theorem 1.

On the other hand, once the desired agreement set $S$ has been computed, the actual construction of an Armstrong relation can be done in time $O(|R| \cdot |S|)$. Therefore we can use for the second part of the algorithm the same construction that was given in [BDFS84]. For completeness, it is repeated below. We have assumed that all domains are integer-valued. Other domains could be substituted without any difficulty.

It is obvious that $S \subseteq \text{agr}(r)$ holds for the constructed relation $r$. The set $\text{agr}(r)$ can well be larger than $S$; the additional elements in $\text{agr}(r)$ are intersections of elements of $S$. As $\text{CL}(F)$ is closed under intersection, and $S \subseteq \text{CL}(F)$, we have $\text{GEN}(F) \subseteq S \subseteq \text{agr}(r) \subseteq \text{CL}(F)$, showing that the construction works correctly.

The dependencies that do not hold in the constructed Armstrong relation can be found by comparing each tuple with the first tuple in the relation. We pointed out in Section 2 that the designer might prefer relations where the observations can be made by looking at consecutive

---

**Algorithm:** Construction of an Armstrong relation for relation scheme $R$ and dependency set $F$.

**Input:** A collection $S$ of subsets of $R$ with $\text{GEN}(F) \subseteq S \subseteq \text{CL}(F)$.

**Output:** A relation $r$ over $R$ such that $S \subseteq \text{agr}(r) \subseteq \text{CL}(F)$.

**Method:**

1. {Initialize}
   - let $r$ be the relation with a single tuple $t$ such that $t[A] = 0$ for all $A \in R$;
   - $i := 1$;

2. {Create the desired $\text{agr}(r)$ set}
   - for each $T \in S$ do begin
     - insert at the end of $r$ a tuple $t$ such that $t[A] = 0$ if $A \in T$,
     - and $t[A] = i$ otherwise;
     - $i := i+1$;
   - end;
tuples. It is possible to achieve this goal without sacrificing the correctness or the efficiency of the algorithm: we can simply replace the condition "\( t[A] = 0 \)" in step 2 of the algorithm by "\( t[A] = s[A] \)", where \( s \) is the previous last tuple in \( r \).

As a final comment about this part of the algorithm, we see that the constructed relation has \(|S| + 1\) tuples. If \( S = GEN(F) \), the discussion at the beginning of Section 3 implies that the size of the generated Armstrong relation is at most (half of) the square of the size of a minimal Armstrong relation.

We now turn to the computation of \( GEN(F) \) or, by Theorem 1, \( MAX(F) \). We have a preliminary result.

**Lemma 2.** Testing whether \( X \in max(F,A) \) can be done in time \( O(|R| + 1 ||F||) \).

**Proof.** Checking that \( X \) does not imply \( A \) takes time \( O(||F||) \); testing for all \( B \in R -X \) that \( XB \rightarrow F \) \( A \) takes time \( O(|R| ||F||) \). \( \square \)

Thus testing for membership in \( MAX(F) \) can be done efficiently. There remains the problem of choosing the candidates for which to apply the test: we do not want to consider all subsets of \( R \).

The solution is to compute \( MAX(F) \) iteratively, starting from a small collection instead of the entire powerset. We start from \( MAX(\emptyset) \). Dependencies are added one at a time and the collection is modified accordingly.

The starting point is easy: \( max(\emptyset,A) = R - \{A\} \) for all \( A \in R \). The following result shows how the sets \( max(G,A) \) relate to the sets \( max(F,A) \) when \( G \) contains one more dependency than \( F \).

**Theorem 4.** Let \( G = F \cup \{Y \rightarrow Y'\} \) and \( W \in max(G,A) \). Either \( W \in max(F,A) \), or for some \( B \in Y \), \( Z \in max(F,B) \) and \( X \in max(F,A) \) we have \( W = X \cap Z \).

**Proof.** Assume \( W \notin max(F,A) \). Since \( W \in max(G,A) \), \( W \not\rightarrow_{F} A \) and \( W \not\rightarrow_{G} A \). Thus some proper superset \( X \) of \( W \) belongs to \( max(F,A) \).
First, $X$ must contain $Y$; otherwise $X^+$ would not change by going from $F$ to $G$. Since $X$ is $F$-closed, this would imply that $X$ is a $G$-closed set that does not determine $A$. But this is impossible, since $W$ is maximal among such sets, and $X$ properly contains $W$.

On the other hand, $Y$ cannot be contained in $W$. For if $Y \subseteq W$, then $YY' \subseteq W$, since $W$ is $G$-closed as a $\text{max}(G,A)$-set. Therefore $YY' \subseteq X$, and $X$ is also $G$-closed. Again $X$ would be a $G$-closed set extending $W$ and not containing $A$, a contradiction.

Thus $Y \setminus W$ is nonempty; let $B \in Y \setminus W$ be arbitrary. The following figure illustrates the relationships of our sets.

![Venn Diagram](image)

Now $W$ cannot determine $B$ in $F$. This follows from the facts that $W$ is both $G$- and $F$-closed as a member of $\text{max}(G,A)$, and that $B \notin W$. Therefore some member of $\text{max}(F,B)$ must contain $W$. Let $Z$ be an arbitrary set in $\text{max}(F,B)$ with this property. We will complete the proof by showing that $W = X \cap Z$.

The first part, $W \subseteq X \cap Z$, follows immediately from the facts that $W \subseteq X$ and $W \subseteq Z$. For the other direction it is sufficient to show that $X \cap Z \not\in F \setminus A$; the membership of $W$ in $\text{max}(G,A)$ then yields that $W = X \cap Z$.

We know that $X \cap Z$ is $F$-closed, since $X$ and $Z$ are. Also $Y \not\subseteq Z$, since $B \in Y$ and $B \notin Z$. Thus $Y \not\subseteq X \cap Z$ and $X \cap Z$ is $G$-closed. Since $X \in \text{max}(F,A)$, we have $A \notin X$ and $A \notin X \cap Z$. Therefore $X \cap Z$ does not determine $A$ in $G$. □

By this theorem, the family $\text{max}(G,A)$ can be computed from $\text{max}(F,A)$ as follows. For each $W \in \text{max}(F,A)$ and each $W$ of the form $X \cap Z$, where $X \in \text{max}(F,A)$ and $Z \in \text{max}(F,B)$ for some $B \in Y$, test whether $W$ is a member of $\text{max}(G,A)$ using the algorithm
of Lemma 2.

This ends our description of generation of reasonably small Armstrong relations. It would be nice to be able to prove that the algorithm works in polynomial time at least with respect to the size of its output. However, for some dependency sets \( G \) it can still happen that the intermediate \( \text{MAX}(F) \) sets for subsets \( F \) of \( G \) are exponential, even though \( \text{MAX}(G) \) is polynomial. As an example, define

\[
F = \{ A_{2i-1}A_{2i} \rightarrow B \mid i = 1, \ldots, m \}, \quad \text{and} \\
G = \{ A_iA_j \rightarrow B \mid 1 \leq i < j \leq 2m \}.
\]

Here \( F \) is the dependency set used in [BDFS84] to prove the exponentiality of \( |\text{GEN}(F)| \). Suppose that we use our algorithm to construct the \( agr \)-set for \( G \) in such a way that the dependencies in \( F \) are considered first (note that \( F \subseteq G \)). Then we will necessarily have an intermediate \( \text{MAX} \)-set whose size is exponential in \( m \). However, it is easy to see that the number of sets in the final \( \text{MAX}(G) \)-collection will only be linear in \( m \). It remains as an open question whether this is an inherent property of the problem, or whether considering the dependencies in some specific order or considering only normalized schemes would yield still better algorithms. Even so, the previous algorithm is more practical than one which considers all subsets of \( R \). Note also that the algorithm is incremental: addition of one dependency can be treated by doing only the computation determined by Theorem 4. This property is useful for the interactive design process.

The problem of minimizing an Armstrong relation was mentioned in Section 2. Using the \( \text{GEN}(F) \) sets it is easily solved: a tuple can be removed from the relation \( r \) if its removal does not make the \( agr(r) \) set too small, i.e. make the condition \( \text{GEN}(F) \subseteq agr(r) \) false.

6. Dependency Inference

In Section 2 we introduced the inverse of the generation problem, the dependency inference problem. Given a relation \( r \), we are asked to produce a set of dependencies \( F \) such that \( r \) satisfies exactly the dependencies in \( F^* \). The exact complexity of this problem is studied in a future paper. Here we will only present the algorithm given below as one nontrivial solution.
Algorithm: Computation of a cover for the set of dependencies satisfied by a relation.

Input: A relation \( r \) over relation scheme \( R \).

Output: A set \( F \) of functional dependencies over \( R \) such that \( r \) satisfies a dependency \( X \to C \) if and only if \( X \to C \in F^* \).

Method:

1. {Initialize}
   - for all \( C \in R \) let \( \text{lhs}(C) = \{\emptyset\} \);
   - \{lhs\( (C) \) is the set of possible left-hand sides \( X \) for dependencies of the form \( X \to C \}\)

2. {Iterate over all pairs of tuples in \( R \)}
   - for all \( t, s \in r, t \neq s \), do
     - for all \( C \in R \) such that \( t[C] \neq s[C] \) do begin
       - update \( \text{lhs}(C) \) with respect to \( t \) and \( s \)
       - for all \( X \in \text{lhs}(C) \) do
         - if \( t[X] = s[X] \) then
           - replace \( X \) in \( \text{lhs}(C) \) by the sets \( X \cup \{D\} \), where \( D \in R \) and \( t[D] \neq s[D] \);
         - while \( \text{lhs}(C) \) contains sets \( X \) and \( Y \) such that \( X \subset Y \) and \( X \neq Y \)
           - do remove \( Y \) from \( \text{lhs}(C) \);
       - end;
     - end;
   - end;

3. {Construct \( F \)}
   - \( F := \{ X \to C \mid C \in R, X \in \text{lhs}(C), \text{ and } C \notin X \} \);

As an example, for the last relation in Section 2 the algorithm would produce the dependency set \{TEACHER HOUR \to ROOM\}.

Theorem 5. Let \( F \) be the dependency set produced for \( r \) by the algorithm above. Then \( r \) satisfies a dependency \( Y \to B \) if and only if \( Y \to B \in F^* \).

Proof. If. Assume \( Y \to B \in F \) (it is enough to prove the claim for members of \( F \) instead of \( F^* \)). If \( r \) does not satisfy \( Y \to B \), then for some \( t, s \in r \) we have \( t[Y] = s[Y] \) and \( t[B] \neq s[B] \). After \( \text{lhs}(B) \) has been updated with respect to \( t \) and \( s \), we have \( W \subseteq Y \) for all \( W \in \text{lhs}(B) \). Since the \( \text{lhs} \)-sets do not decrease, \( Y \) cannot belong to \( \text{lhs}(B) \) at the end of the algorithm. Therefore \( Y \to B \notin F \), contradicting the assumption.

Only if. Assume \( r \) satisfies \( Y \to B \). We claim that after the execution of the algorithm,

\[
Z \in \text{lhs}(B) \text{ for some } Z \subseteq Y . \tag{1}
\]

This holds trivially after the initialization. Consider then the updating of \( \text{lhs}(B) \) on the basis of \( t \)
and \( s \). Suppose that \( Z \subseteq Y \) belongs to \( \text{lhs}(B) \) before the modification. If \( t[Z] \neq s[Z] \), \( Z \) remains in \( \text{lhs}(B) \), and (1) holds after the update as well. Suppose then that \( t[Z] = s[Z] \). Since \( r \) satisfies \( Y \rightarrow B \) and \( t[B] \neq s[B] \), we must have \( t[Y] \neq s[Y] \). In particular, for some \( D \in Y - \{Z\} \) we have \( t[D] \neq s[D] \). But then \( Z \cup \{D\} \) is added to \( \text{lhs}(B) \), and (1) still holds.

We must still consider the "cleaning" of \( \text{lhs}(B) \), i.e. the \texttt{while}-loop at the end of step 2 of the algorithm. It is easily seen that in neither of the two cases of the previous paragraph can the crucial set be removed from \( \text{lhs}(B) \) by the cleanup. Therefore (1) holds at the end of the execution of the algorithm. But this means that \( Z \rightarrow B \in F \), implying \( Y \rightarrow B \in F^* \). \( \square \)

7. Concluding remarks

We have studied the problem of generating small Armstrong relations for database design. We started by analyzing the use of these relations when designing relation schemes, continued by approximating the sizes of the relations for normalized schemes, and concluded by giving algorithms for generating reasonably small Armstrong relations and for inferring a set of dependencies from a relation.

There are many interesting open problems in this field. The complexity of the dependency inference problem was already mentioned. Another question that we are currently studying is the production of truly minimum Armstrong relations - recall that our algorithm may produce a relation whose size in the worst case can be proportional to the square of that of a minimum relation. Finally, the implementation that we are carrying out should shed some light on practical questions, such as the most convenient structure of an Armstrong relation.

A whole new area opens when one includes other types of dependencies, not only functional dependencies. An attractive addition would be to consider inclusion dependencies; for them and functional dependencies Armstrong databases do exist [FV83]. For this class Armstrong databases can be especially useful, since they can show the amount of duplication caused by the inclusion dependencies. Research into this area would also have to clarify the basic ideas for designing relation schemes when inclusion dependencies are present. A preliminary attempt at such a method
is presented in [MR85].

References


