A method for computing the closure of a set of attributes according to a specification of Functional Dependencies of the relational model is shown. The main feature of this method is that it computes the closure using solely the inference system of the S\textsubscript{L}\textsubscript{FD} logic. For the first time, FD Logic is used in the design of automated deduction methods to solve the closure problem. The strong link between S\textsubscript{L}\textsubscript{FD} logic and the closure algorithm is presented and an S\textsubscript{L}\textsubscript{FD} simplification paradigm emerges as the key element of our method. In addition, the soundness and completeness of the closure algorithm is shown. Our method has the same complexity, in the worst case, as the classical closure algorithms and it has all the advantages provided by the use of logic. We have also empirically compared our algorithm with the Diederich and Milton classical algorithm. This experiment reveals the best behavior of our method which shows a significant improvement in the average speed.

Keywords: Logic, Closure operator, Functional dependencies.

AMS Subject Classification: 03B80; 68P15; H.2.0.

Preliminary discussion: about the Functional Dependency notion

In this section we present a brief description of the Theory of Relations and Functional Dependencies (FDs) as well as a wide range of areas in which the notion of FD has been shown to be useful. A survey of this concept can be seen in [42].

FDs were introduced by William W. Armstrong in [3]. These dependencies are constraints which represent relationships between data and describe data knowledge. Their success was mainly due to the development of the Theory of Normalization, which provides a formal framework to characterize efficient relational databases [12]. Nevertheless, this success has relegated to a secondary role the more recent development of theories and methods promoting the use of FDs in a wide variety of areas.

The notion of data dependency used in these areas is basically the same and consequently, the methods designed to reason about FD can be swapped.

Functional dependencies arise frequently in pure declarative specifications, because of the intermediate results that need to be computed in order to express some of the constraints, or as a result of precise modeling choices, e.g., to provide multiple viewpoints of the search space in order to increase propagation. Either way, the dependencies recognition greatly helps solvers, allowing them to avoid spending time on unfruitful branches, while maintaining the highest degree of declarativeness. In [8], by modeling constraint problem specifications as second order, the
authors provide a characterization of functional dependencies in terms of semantic properties of first-order ones.

Another area in which the concept of functional dependency has been proven useful is logic programming. Prolog systems provide a control primitive called cut that gives the programmer some say in how backtracking is handled. However, the use of the cut or other extra-logical control operators makes programs less declarative and more procedural, partially defeating the goals of logic programming. In [39], the author proposes to use declarative information provided by the programmer in the form of functional dependencies to automate the insertions of cuts.

The use of functional dependencies to enrich the specification is in itself valuable, but it is even more so when the management of the enriched specification renders an improvement in the performance of these systems. The work presented in [49] is a very illustrative example of the good use of FD specifications. The authors focus on the problem of grid resource management: to schedule and plan a set of finite resources (CPU utilization, network load, storage allocation, etc) that are shared by a large set of complex systems composed of thousands of components from disjointed domains. Normally, resource management applied in grid computing is used for a single application, but the authors cited focus on solving the resource management for dynamic and complex emerging applications. The application components are modeled with resource requirements and functional dependencies, stored in the dependency repository. The authors have designed a scheduling algorithm to distribute the application request in grid computing which improves the efficiency when it uses the FD repository. They have compared the method applied to a set of experiments with and without functional dependencies, with the following conclusion: “these results have proved that our system gains better performance when applying functional dependencies”.

In the context of declarative programming, in [43] the author considers the problem of having an extension of functional languages to support multiple parameters, allowing a more general interpretation of classes as relation of types, which would provide many potentially useful applications. The problem is addressed by adding FDs to the specification. In [18] the popularity and usefulness of this extension was observed: “Functional Dependencies introduced by M. Jones has proved to be a very attractive extension to multi-parameter type classes in Haskell”. In this recent paper, the soundness and decidability of an inference type for FDs was proven.

The Functional Dependency concept is also useful for the design of Neuronal Networks and it can improve the efficiency of these networks that model the behavior of microwaves. Classical multilayer perceptrons (MLP) have been used for this purpose. However, in [44], the authors show the disadvantages of these solutions and propose a new knowledge-based neural (KBN) model for microwave modeling to overcome these problems. KBN networks are neural networks which incorporate existing partial knowledge (already known functional dependencies) about modeling problems.

Functional dependencies were recently studied in the context of knowledge representation [27]. In [28] a theory is viewed as a relation, and then a functional dependency expresses the fact that, in all models of the theory, the value of a variable is determined by the values of some other variables.

Another use of FDs is found in the design of classifier systems. For example, in [13], the author presents an FD based expert system which automatically translates the different ways of representing knowledge in the classifier systems (production rules, decision tables and decision trees). In this expert system, the set of FDs can also be used to filter cases to be classified, eliminating spurious cases and cases for which the classification is likely to be of doubtful validity.
Ontologies have emerged in the Semantica Web field to describe terms, retrieve information and interconnect Web services. In [6] the authors use FDs to derive a normalized logical schema of a database for a ontology-based databases (OBDB). The theory of ontology translation and integration over arbitrary logics is explored in [21].

Concerning Software Engineering, despite the use in the Relational Database, the notion of FDs has been used successfully to specify several constraints on the data stored in the web. In [24], the authors remark that “The importance of XML integrity constraints is due to a wide range of applications ranging from schema design, query optimization, efficient storing and updating, data exchange and integration, to data cleaning”. In this area, the authors have put a lot of effort into the development of efficient methods to manage FDs: [2, 7, 20, 31, 34, 52] We would like to point out that most of these algorithms are not based directly on a logic inference system, which limits their applicability to other areas beyond the Information Systems.

In addition, some papers show the application of FDs in other areas. It is beyond the scope of this paper to provide an exhaustive list of these applications. As an interesting example, FDs have been used to solve some hardware problems and applied to the study of relaying coordination in the framework of relay settings in a protection scheme. In this field FDs have the following definition: “The settings of the relays are viewed as the attributes of the protection scheme, and the functional dependencies are generated from the primary-backup pairs.”[29]

1. Introduction

In the previous section, we have illustrated the need to work with a general definition of the notion of functional dependency. Thus, here we present this concept using definitions close to the theory of relations.

Given a finite set of indexes $A$ whose elements will be named attributes and a family of sets $\{D_i \mid i \in A\}$, in the class of the n-aries relations, $\mathcal{C} = \{R \subseteq \prod_{i \in A} D_i\}$, a functional dependency (FD) is an expression of the type $X \rightarrow Y$, where $X, Y \subseteq A$. It is read as “$X$ determines $Y$”.

In a logic style, an FD is an expression which determines a subset of models, where each one is a relation which satisfies the FD. Formally, the class of models of $X \rightarrow Y$, $\mathcal{C}_{X \rightarrow Y}$, is the set of $R \in \mathcal{C}$ such that, for all $(t_1, \ldots, t_n)$, $(t'_1, \ldots, t'_n) \in R$, if $t_i = t'_i$ for all $i \in X$ then $t_i = t'_i$ for all $i \in Y$.

For a set of functional dependencies $\Gamma$, the class of models of $\Gamma$ is

$$\mathcal{C}_\Gamma = \bigcap_{X \rightarrow Y \in \Gamma} \mathcal{C}_{X \rightarrow Y}$$

Functional dependencies are fundamental constraints which represent relationships between attribute sets [12] and they are considered as constraints to describe data knowledge. Although FDs may be defined in terms of partial injective functions, W.W. Armstrong [3] introduced the so called Armstrong Axioms, which formally state the FD semantics. He presents these semantics by introducing a logic inference system:

**Proposition 1.1 Armstrong Axioms**

Let $X, Y, Z$ be arbitrary subsets of the set $A$ of attributes. We agree that $XY$ stands for the union of $X$ and $Y$. The following FD inference system is sound and complete:
Closure via FD Simplification

(1) If $Y \subseteq X$ then $X \rightarrow Y$. \hspace{1cm} \text{(Axiom)}

(2) If $X \rightarrow Y$ then $XZ \rightarrow YZ$. \hspace{1cm} \text{(Augmentation Rule)}

(3) If $X \rightarrow Y, Y \rightarrow Z$ then $X \rightarrow Z$. \hspace{1cm} \text{(Transitivity Rule)}

This paper proposes for the use of logic for dealing with Functional Dependencies. This issue is not new; in the literature several functional dependencies logics exist, which follow the Armstrong Axioms [4, 19, 27, 42, 51]. All these logics are frequently used in the natural deduction-like tasks, but none of them have been used successfully in an automated deduction style. The reason is that their corresponding inference systems were created to explain dependency semantics more than to design an executable automated deduction system. All these logics are mainly based on the Transitivity Rule, which is not suitable for direct application in a deduction method.

Efficient methods to manage functional dependencies are needed to improve and automatize the normalization process. Thalheim [46] considers that, despite the great number of existing publications on this subject, there still is a lot to do. In [4, 10, 27, 33, 37, 45], indirect methods for the use of dependencies are used and the difficulty of managing FDs from logic is discussed in [22, 23, 27].

The algorithms dealing with FDs that appear in the literature use exhaustively the closure operator of a set of attributes. For any subset $X$, of a given set $U$ of attributes, the closure of $X$ is the maximum (for the inclusion) set $Y \subseteq U$ such that $X \rightarrow Y$ holds, w.r.t. the Armstrong axioms. This set is denoted by $X^+$. In the literature there are several algorithms to compute the closure of a set of attributes in linear time (see [4, 17] for further details). All these efficient methods take as input $X$ (a set of attributes), $\Gamma$ (a set of FDs) and, by crossing the dependencies of $\Gamma$, they return $X^+$. These methods obtain such a low cost because they use data structures to avoid the use of each functional dependence more than once.

Notice that the closure problem goes beyond the scope of FDs. As N. Casspad and B. Monjardet cite in [9]: “Closure systems or equivalently, closure operators and full implicational systems appear in many fields in pure or applied mathematics and computer science”. In this paper they list the numerous domains where closure systems and closure operators are successfully applied: algebra, topology, geometry, logic, combinatorics, computer science, data analysis, knowledge structures, mathematics of social science, etc. In [53] M. Wild remarks that in many applications the closure algorithm appears as a subroutine: database theory, formal concept analysis, matroid theory, algebra, etc.

Moreover, the computation of $X^+$ is abundantly used in the database literature and is one of the key points in many problems: redundant dependency elimination, query optimization, the key-finding problem, etc. Specifically, the most frequently used algorithms to transform relational databases into more efficient ones, use exhaustively the closure of a set of attributes. (see[16, 17, 36, 55])

In [1, 14] we present a new logic, named $\text{SL}_{FD}$ (Substitution Logic for FDs), which includes two new substitution rules suitable for automated deduction. Substitution Logic was formally introduced in [14], providing soundness and completeness results.

This logic is based on a formal study of the concept of functional dependency within the general framework of the lattice theory [15]. Its novel inference system has opened the door to the design of efficient algorithms for the management of FDs: to remove redundancy [41] and to solve the implication problem [1]. This paper represents another step forward in the use of the $\text{SL}_{FD}$ logic.

In [40] we showed a comparative study between classical closure algorithms and a closure algorithm based on $\text{SL}_{FD}$ logic. The main goal of that paper was to collect the best known classical closure algorithms to compare them with our method.
The result of that empirical study was the best behaviour of our algorithm. In the other hand, among the classical methods, the best algorithm was the Diederich and Milton method.

In this paper we present a new version of the closure algorithm to obtain the closure \( X^+ \) of a set \( X \) w.r.t. a set of FDs. This new version is directly related with three equivalences deduced from the substitution rules of \( \text{SL}_{FD} \) logic. FD logic is the only tool used to develop the method using an automated deduction style. The substitution rules were designed to simplify sets of FDs and in this work we show how they can be used to compute the closure. The three equivalences presented in this paper propose three FD simplification patterns and they are an important factor in the efficiency of our algorithm.

In this work, the complexity, the soundness and completeness of the closure algorithm is presented. Our algorithm has the same complexity (in the worst case) as the classical efficient closure algorithms. We have also designed an empirical study which shows that our method has a significant speed improvement on average when compared to the Diederich and Milton classical method. As previously mentioned, we have selected the Diederich and Milton algorithm because of its good behaviour in the preliminary experiment presented in [40]. Here, we have designed another experiment increasing the number of FDs and we have considered two separate cases depending on the ratio of attributes in the FDs. As a general conclusion, from this study we infer that our method is significantly faster than the classical Diederich and Milton method in both cases.

2. Background

A closure algorithm for a set of attributes was proposed in [35, 50] (see function 1 below). This algorithm is the first one to appear in the literature to compute the closure of a set of attributes. Consider \( U \) to be a set of attributes and \( \Gamma \) to be a set of FDs, then its complexity is \( O(|U| |\Gamma|^2) \) in the worst case.\(^1\) A complexity study of this algorithm is available in [17].

<table>
<thead>
<tr>
<th>Function 1: Standard Closure(( U, \Gamma, X ))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output:</strong> ( X^+ )</td>
</tr>
<tr>
<td><strong>begin</strong></td>
</tr>
<tr>
<td>( X^+ = X )</td>
</tr>
<tr>
<td><strong>repeat</strong></td>
</tr>
<tr>
<td><strong>foreach</strong> ( A \rightarrow B \in \Gamma ) <strong>do</strong></td>
</tr>
<tr>
<td><strong>if</strong> ( A \subseteq X^+ ) and ( B \notin X^+ ) <strong>then</strong></td>
</tr>
<tr>
<td>( X^+ := X^+ \cup {B} )</td>
</tr>
<tr>
<td><strong>end</strong></td>
</tr>
<tr>
<td><strong>end</strong></td>
</tr>
<tr>
<td><strong>until</strong> no more attributes are added to ( X^+ )</td>
</tr>
<tr>
<td><strong>Return</strong> ( X^+ )</td>
</tr>
</tbody>
</table>

In [5, 17, 42] several algorithms are presented, which compute in linear time closures. An empirical study of the behaviour of these algorithms is presented.

\(^1\)As usual \(|X|\) denotes the cardinality of the set \( X \).
in [40] and an analysis of the strategies used in those algorithms was also included. All of them use some data structures to reduce the cost of traversing the sets $\Gamma$ (FDs) and $U$ (attributes), decreasing the cost of the $X^+$ computation. The most common reduction strategies are the following:

(a) To use a set to keep track of those attributes that still have to be added to the closure.

(b) To use an array indexed by the atomic attributes $A_i$ to keep track of the FDs that have the attribute $A_i$ in the left hand side of the FD.

(c) To keep track of the number of attributes belonging the left-hand side for each FD that are not yet in the closure.

One conclusion of the empirical study presented in [40] was that the most efficient classical closure algorithm is Function 2 [17] below and developed by Diederich and Milton. This algorithm uses the following data structures to improve the complexity of the standard closure algorithm:

- The authors use $UPDATE$ to apply strategy (a).
- The authors use $LIST[A_i]$ to apply strategy (b).
- The authors use $COUNT[A \rightarrow B]$ to apply strategy (c).

Function 2: Linear Closure($U$, $\Gamma$, $X$)

Output: $X^+$

begin
  $X^+ := X$
  $UPDATE := X$
  foreach $A \rightarrow B$ do
    $COUNT[A \rightarrow B] := |A|$
  end
  foreach attribute $A_i$ do
    $LIST[A_i]$ is a list of pointers to those dependencies in which $A_i \in A$
  end
  while $UPDATE$ is not empty do
    Select and remove an attribute $A_i$ from $UPDATE$
    foreach $A \rightarrow B \in LIST[A_i]$ do
      decrement $COUNT[A \rightarrow B]$
      if $COUNT[A \rightarrow B] = 0$ then
        add $B$ to $UPDATE$ and add $B$ to $X^+$
        if $B$ is not already in $X^+$
      end
    end
  end
end

In [42], the authors show that the complexity of this algorithm is $O(|U| |\Gamma|)$. They also mention that “in the literature $O(|U| |\Gamma|)$ is usually considered as the order of the input. From this point of view, this is a linear time algorithm for the computation of the closure of a set of attributes”.
3. The SL\textsubscript{FD}-closure algorithm

As we mentioned in section 1, for the first time, we use an FD logic to develop automated deduction methods for closure computation. In this section, we summarize the axiomatic system SL\textsubscript{FD} [14]. It is guided by the idea of simplifying the set of FDs by removing redundant attributes efficiently. This is one of the characteristics of SL\textsubscript{FD} logic because other well-known FD logic systems are guided by Armstrong Relations [3], more oriented to capturing all the FDs which can be deduced from a given set of FDs.

Another important characteristic of SL\textsubscript{FD} logic is the definition of two substitution rules which have not been defined in other FD logics.

There, we introduce formally the syntax of SL\textsubscript{FD} logic.

**Definition 3.1** The SL\textsubscript{FD} language

Let \( \mathcal{U} \) be an infinite enumerable set of atoms (attributes) and let \( \rightarrow \) be a binary connective.

The language of SL\textsubscript{FD} logic, named \( \mathcal{L}_{\text{FD}} \), is the set of the following well formed formulae \( ^1 \).

\[
\mathcal{L}_{\text{FD}} = \{ X \rightarrow Y \mid X, Y \in 2^{\mathcal{U}} \}
\]

**Definition 3.2** We define the SL\textsubscript{FD} logic as the pair \( (\mathcal{L}_{\text{FD}}, S_{\text{FD}}) \) where \( S_{\text{FD}} \) has one axiom scheme:

\[
[\text{Ax}_{\text{FD}}] : \vdash X \rightarrow Y, \text{ where } Y \subseteq X
\]

Specifically, \( X \rightarrow \top \) is an axiom scheme.

The inference rules are the following:

- **Fragmentation rule**
  \[
  [\text{Frag}] \quad X \rightarrow Y \vdash X \rightarrow Y', \text{ if } Y' \subseteq Y
  \]

- **Composition rule**
  \[
  [\text{Comp}] \quad X \rightarrow Y, U \rightarrow V \vdash XU \rightarrow YV
  \]

- **Substitution rule**
  \[
  [\text{Subst}] \quad X \rightarrow Y, U \rightarrow V \vdash (U - Y) \rightarrow (V - Y)
  \text{ if } X \subseteq U, \: X \cap Y = \emptyset
  \]

In [14] we prove the equivalence between SL\textsubscript{FD} logic and the classical FD logics and we point out the advantages of the first one. This equivalence result is not trivial because our logic does not have the transitivity as a primitive rule, but a derived rule. We also present another useful derived inference rule:

**r-Substitution Rule**

\[
[r\text{Subst}] \quad X \rightarrow Y, U \rightarrow V \vdash U \rightarrow (V - Y)
\text{ if } X \subseteq UV \text{ and } X \cap Y = \emptyset
\]

The following formula scheme is used in our approach: \( X \rightarrow \top \). In [48] the author considers this FD scheme to solve some problems, concerning the management of FDs in a given algorithm but \( X \rightarrow \top \) does not appear in any FD logic in the literature. The symbol \( \top \) denotes the empty set of attributes and it is used in our

\( ^1 \)As usual, \( XY \) is used as the union of sets \( X, Y \); \( X \subseteq Y \) as \( X \) included in \( Y \); \( Y - X \) as the set of elements in \( Y \) that are not in \( X \) (difference) and \( \top \) as the empty set.
Closure via FD Simplification

approach to compute the closure of $X$ by introducing the expression $\top \rightarrow X$ as a seed in the execution.

The following theorem states that both, substitution and r-substitution rules may be used to define transformations of FD sets, which preserves equivalence.

**Theorem 3.3** Consider $X \rightarrow Y, U \rightarrow V \in \mathcal{L}_{FD}$ with $X \cap Y = \emptyset$.

(a) If $X \subseteq U$ then

$$\{X \rightarrow Y, U \rightarrow V\} \equiv \{X \rightarrow Y, U-\top \rightarrow V-\top\}$$

(b) If $X \subseteq UV$ then

$$\{X \rightarrow Y, U \rightarrow V\} \equiv \{X \rightarrow Y, U \rightarrow V-\top\}$$

The proof of the above theorem and an illustration of how $\mathbf{SL}_{FD}$ logic directly removes redundancy in a set of FDs can be seen in [14]. In [41] we carry out an empirical study to prove the practical benefits of the substitution rules.

<table>
<thead>
<tr>
<th>Function 3: $\mathbf{SL}_{FD} - \text{closure}(U, \Gamma, X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output:</strong> $X^+$</td>
</tr>
<tr>
<td><strong>begin</strong></td>
</tr>
<tr>
<td>$\Gamma' := \Gamma \cup {\top \rightarrow X}$</td>
</tr>
<tr>
<td>$X_{new} := X$</td>
</tr>
<tr>
<td>$X_{old} := X$</td>
</tr>
<tr>
<td><strong>repeat</strong></td>
</tr>
<tr>
<td>Replace ${\top \rightarrow X_{old}}$ with ${\top \rightarrow X_{new}}$ in $\Gamma$</td>
</tr>
<tr>
<td>$X_{old} := X_{new}$</td>
</tr>
<tr>
<td><strong>foreach</strong> $A \rightarrow B \in \Gamma' - {\top \rightarrow X_{new}}$ <strong>do</strong></td>
</tr>
<tr>
<td>Reduce in $\Gamma'$ from ${\top \rightarrow X_{new}, A \rightarrow B}$ to</td>
</tr>
<tr>
<td>$\begin{cases}</td>
</tr>
</tbody>
</table>
| \{\top \rightarrow X_{new}B\} : X_{new} := X_{new}B^\ast & \text{if } A \subseteq X_{new} \\
| \{\top \rightarrow X_{new}\} & \text{if } B \subseteq X_{new} \\
| \{\top \rightarrow X_{new}, (A - X_{new}) \rightarrow (B - X_{new})\} & \text{if } A \cap X_{new} \neq \emptyset \text{ or } B \cap X_{new} \neq \emptyset |
| $\end{cases}$                                              |
| **end**                                                     |
| **until** $X_{new} = X_{old}$                               |
| Return $X^+$                                                |
| **end**                                                     |

/* * In this step, we replace $\{\top \rightarrow X_{new}, A \rightarrow B\}$ with $\{\top \rightarrow X_{new}B\}$ and assign $X_{new}B$ to $X_{new}$ */

Function 3 is a version of the method presented in [40]. This version is based on the strong link between the simplifications carried out by the algorithm and a set of equivalences inferred from the $\mathbf{SL}_{FD}$ axiomatic system. It is the first time where a closure algorithm is directly based on an FD inference system, providing an automated deduction style.

In order to obtain $X^+$, we present here a technique to simplify the input set of FDs in a systematic way. The simplification is based on three new equivalence patterns which reduces the FD specification. It is surprising that the systematic use of these simplification equivalences computes the closure. The equivalences are presented as follows:
• **Equivalence I**: If \( U \subseteq W \) then
\[
\{ \top \rightarrow W, U \rightarrow V \} \equiv_{\text{SLFD}} \{ \top \rightarrow WV \}
\]

• **Equivalence II**: If \( V \subseteq W \) then
\[
\{ \top \rightarrow W, U \rightarrow V \} \equiv_{\text{SLFD}} \{ \top \rightarrow W \}
\]

• **Equivalence III**: If \( U \cap W \neq \emptyset \) or \( V \cap W \neq \emptyset \) then
\[
\{ \top \rightarrow W, U \rightarrow V \} \equiv_{\text{SLFD}} \{ \top \rightarrow W, U - W \rightarrow V - W \}
\]

Given a set of FDs \( \Gamma \) and a set of attributes \( X \), the execution of the method is triggered by adding the seed \( \top \rightarrow X \) to \( \Gamma \), rendering an initial \( \Gamma' \). The three equivalences are then systematically applied to the FD set. Finally, when none of the three equivalences can be applied, in \( \Gamma' \) there exists a unique FD of the type \( \top \rightarrow Y \) and \( Y \) is precisely \( X^+ \). We would like to point out that the algorithm is driven by the expression \( \top \rightarrow X \).

The theoretical justification of equivalences I, II and III is given below:

• **Equivalence I**:
\[
\{ \top \rightarrow W, U \rightarrow V \} \overset{1}{=} \{ \top \rightarrow W, U - W \rightarrow V - W \} \overset{2}{=} \{ \top \rightarrow WV \}
\]

The substitution rule is applied in 1 and, since \( U \subseteq W \), the composition rule is applied in 2.

• **Equivalence II**:
\[
\{ \top \rightarrow W, U \rightarrow V \} \overset{3}{=} \{ \top \rightarrow W, U - W \rightarrow V - W \} \overset{4}{=} \{ \top \rightarrow W \}
\]

The substitution rule is applied in 3 and, since \( V \subseteq W \), the Axiom is applied in 4.

• **Equivalence III**:
\[
\{ \top \rightarrow W, U \rightarrow V \} \overset{5}{=} \{ \top \rightarrow W, U - W \rightarrow V - W \}
\]

The substitution rule is applied in 5.

This theoretical justification ensures that if we apply equivalences I, II or III to an input \( \Gamma \cup \{ \top \rightarrow X \} \) and we obtain \( \Gamma' \cup \{ \top \rightarrow Y \} \), then we have that \( \Gamma \vdash X \rightarrow Y \). On the other hand, it is easy to prove that if \( Y \not\subseteq X^+ \), we can jet apply equivalence I, II or III. Finally, the algorithm always ends because the size of the set of FDs is strictly decreased. This ensures the soundness and completeness of our method.

**Example 3.4** Let \( \Gamma \) be the following set of dependencies.
\[
\{ ak \rightarrow bc, cd \rightarrow gh, cij \rightarrow kl, de \rightarrow f, g \rightarrow de, hf \rightarrow ia, f \rightarrow c \}
\]
In order to compute \((afd)^+\), we first define the input \(\Gamma' = \Gamma \cup \{\top \rightarrow afd\}\). The algorithm renders

\[
\{ \top \rightarrow afdcgehi, k \rightarrow b, j \rightarrow kl \}
\]

and, therefore, the closure of \((afd)\) is

\[
(afd)^+ = afdcgehi
\]

Figure 1 shows step by step the application of the \(\text{SL}_{FD}\) closure algorithm. We depicted the iterations of the \(\text{Repeat}\) loop in rows. We also label each FD with the applied Equivalence (I,II,III or none). We use the symbol \(\times\), if the FD is removed from \(\Gamma\). Moreover, we illustrate the increase of the expression \(\top \rightarrow X_{new}\) when Equivalence I is applied.

Regarding complexity results, our method is as good as other efficient methods that appear in the literature. In the worst case, the \(\text{Repeat}\) loop (label [1] of the algorithm) is executed at most \(|U|\) times, since in every iteration at least one attribute is added to \(X_{new}\). The \(\text{For}\) loop (label [2] of the algorithm) is executed at most \(|\Gamma|\) times. Consequently, the complexity of the algorithm is \(O(|U| |\Gamma|)\). Our algorithm has the same complexity (in the worst case) as the previous algorithms cited in the literature, namely linear with regard to the input.

4. Conclusions and statistics of the application of closure algorithms

In [14] we have proposed a new FD logic named \(\text{Substitution Logic for functional dependencies (SL}_{FD}\) logic) and we illustrate how \(\text{SL}_{FD}\) logic directly removes redundancy in a set of FDs. In [41] we propose a more complete preprocessing transformation based on \(\text{SL}_{FD}\) logic and we show an empirical study analyzing the benefits of the new substitution rules incorporated into \(\text{SL}_{FD}\) logic. In [1] we prove that it is possible to use the paradigm of rewriting systems to remove redundancy in FD sets and we use the Maude rewriting language to execute directly the rules of \(\text{SL}_{FD}\) logic using the Maude environment [11].

In this paper, the closure algorithm with a formal base, the \(\text{SL}_{FD}\) logic, is proposed. The complexity in the worst case is the same as that in the classical linear closure algorithms. Furthermore, in this work we have carried out an empirical study about the execution time \(^1\) of Algorithm 2 and Algorithm 3. To do so we have implemented them using \(C++\).

This empirical study extends the one presented in [40]. Here we have increased the number of FDs and we have also considered two separate situations, explained.

---

\(^1\)In [53] the author remarks that ‘the execution times are mostly relevant for comparing algorithms’. 
later in this section. Here, we have only compared our method with Diederich and Milton method, because in [40] we show that it is the best candidate among the well known classical methods.

To improve random characteristics of the random number generator used in the empirical study, we have not used the usual random library of C++ and we have implemented the algorithm \textit{ran3}. This algorithm is shown in [54] following Knuth’s suggestion and it is a portable routine based on a subtractive method.

In order to have another criterion measurement, we use another parameter defined in [4, 53]. The authors define the size of an FD set as follows:

\textbf{Definition 4.1.} Let $\Gamma = \{X_1 \rightarrow Y_1, \ldots, X_n \rightarrow Y_n\}$ a set of FDs, we define the size of $\Gamma$ as $||\Gamma|| = \sum_{i=1}^{n}(|X_i| + |Y_i|)$.

We randomly generate a set of attributes $U$, a set of FDs $\Gamma$, and a subset of attributes $X \subseteq U$ and we apply both closure computing methods. The software renders four data: the cardinality of the random FD set, the size of the random FD set, the execution time of the classical closure and the execution time of the SLFD closure.

In this study, we have generated FD sets with 100, 200 and 300 FDs and we have performed two experiments with the ratio 1/4 and 3/4 between the maximum number of attributes and the cardinality of the FD set. The size of the random FD sets is in the range between 2337 and 46870 attributes\(^1\). In figure 2 we depicted

\(^1\)A CSV file with the results, the source code of the algorithms and the executable version of them for
the results of the experiment, where the x-axis represents the time in milliseconds and the y-axis represents the size of $\Gamma$.

This figure shows that $\text{SL}_{FD}$ Closure (the dashed curve) has had a significatively better behavior than classical Linear Closure (bottom curve). In 594 cases out of 600, that is in 99 per cent of the cases, $\text{SL}_{FD}$ Closure has calculated the attributes closure using less time than Linear Closure. The average time using $\text{SL}_{FD}$ Closure is 110.86 and using Linear Closure it is 942.05. We have used the t-test and Wilcoxon-test to compare times used by the two methods and both tests show that the probability of the observed result being due to chance (the mean of $\text{SL}_{FD}$ Closure is lower than mean of Linear Closure) is less than 0.0005.

Acknowledgments

This paper has been partially supported by projects TIN2007-65819 (Science Ministry of Spain) and P06-FQM-02049 (Junta de Andalucía).

References

REFERENCES


