A new closure algorithm based in logic: $\mathbf{SL}_{FD}$-Closure versus classical closures.[*]

A. Mora, G. Aguilera, M. Enciso, P. Cordero, I.P. de Guzmán

E.T.S.I. Informática.
Universidad de Málaga
Campus de Teatinos s/n.
Málaga, 29071, Spain
formethis@ctima.uma.es

Resumen

The field of application of closure systems goes from theoretical areas as algebra or geometry to practical areas as databases and artificial intelligence. In these practical areas, a kind of constraint named functional dependencies have an important role. Given a set of attributes X and a set of functional dependencies Γ, the computation of the closure of X for Γ, denoted as X^+ is abundantly used in artificial intelligence and database literature and is one of the key points in many problems: knowledge compilation, redundant constraint elimination, query optimization, the finding key problem, etc. We outline the main classical closure algorithms and we compare them with a novel algorithm named $\mathbf{SL}_{FD}$-Closure. We show an empirical study with the execution of the closure algorithms, and we establish that $\mathbf{SL}_{FD}$-Closure is the fastest.

Palabras clave: Closure, Logic, Functional Dependencies.

1 Introduction

We illustrate the background work related with the constraint named functional dependency well known in artificial intelligence (AI) and in relational database. In [17] the authors study a condensation knowledge procedure: "the knowledge of functional dependencies in a theory may allow to simplify the theory by eliminating those variables whose values are determined by the values of other values". When defining a database by using the relational model (see [26]), it is necessary to establish a set of dependencies (constraints) to represent data semantics.

Thalheim, in [29], states that, nowadays, there exist more than 100 kinds of dependencies in the literature on databases and he considers that, despite the great number of existing publications on this subject, there still is a lot to do.

Recently, there exists a wide range of problems in database which are being treated successfully with AI techniques.Thus, [7] pursue the integration between database and AI techniques, in [15] non classical logics are applied to *specification* and *verification* of programs, [28] shows the useful characteristics of logic for Information Systems, etc.

Moreover, in [14] the authors emphasize that the

[*]This paper has been partially supported by the Spanish project TIC03-8687-C02-01.

solution to classical problems in database theory can provide important support in underpinning the reasoning and learning applications encountered in AI.

More recently, some authors have related functional dependencies with emergent technologies. Thus, the extension of functional dependencies to XML has been studied in [2, 8, 19] and the extraction of functional dependencies from large databases has been performed using data mining in [6, 21, 23].

Other authors remark in their works the importance of functional dependencies. Thus, in [6] the authors mention the need to discover knowledge which is latent in database relations and the possibility of "*checking dependencies and finding keys for a conventional relation with a view to using the solution in general knowledge discovery*". In [19] it is stated that "*this constraint can help users to identify semantically incorrect transformations*".

Now, we center our approach. In the literature of AI and database appears a great amount of works [16, 18, 32, 33] related with the implication problem. In order to solve this problem is common to use the closure algorithm to obtain the closure X^+ of a set X w.r.t. a set of functional dependencies.

As N. Caspard 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 the paper the author enumerates the great quantity of domains in which 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 [24] we present a new closure algorithm named $\mathbf{SL}_{FD}$-Closure with the same complexity in the worst case that the linear closure algorithms. This algorithm applies the rules of $\mathbf{SL}_{FD}$ logic [11] and it has the same complexity of the classical efficient closure algorithms. An FD logic is used to develop an automated deduction method which computes the closure operator. In this work, we present a new version of the $\mathbf{SL}_{FD}$-Closure algorithm and we show an empirical study about the new algorithm to check what is the faster closure algorithm.

In section 2, we summarize the basic concepts for functional dependencies and we remember what is the implication problem for functional dependencies. We outline the main strategies used in the literature to improve closure algorithms in section 3. We remark the novel closure algorithm in section 4. In section 5, we present the empirical study that we have developed and section 6 contains the conclusions and outlines the actual and future works. Section 7 show the algorithms that we have implemented in C++ for the empirical study.

2 Preliminaries

In this paper, we focus on functional dependencies, introduced by E. F. Codd [10] in 1970. Functional dependencies may be considered as the reinforced structure of the building designed by E.F. Codd: the database Relational Model. They are used to determine good design criteria (normalization theory), to enhance database systems performance (query optimization), to describe and to infer data knowledge (data mining), etc.

Definition 2.1 (Functional Dependency)

Let R be a relation over a set of attributes $\mathcal{U}$.

- *Any affirmation of the type $X \mapsto Y$, where $X, Y \subseteq \mathcal{U}$, is named* **functional dependency** *(henceforth* **FD***) over R*
- *We say that R* **satisfies** *$X \mapsto Y$ if, for all $t_1, t_2 \in R$, we have that:*

$$t_{1/X} = t_{2/X} \text{ implies that } t_{1/Y} = t_{2/Y}$$

When we introduce a relation, we must specify the set of FDs that are satisfied by this relation. Thus, we introduce the following definition:

Definition 2.2 (Relation Scheme) *Let $\mathcal{U}$ be a set of attributes. We say that $\mathcal{S} = (R, \Gamma)$ is a* **relation scheme** *if R is a relation over $\mathcal{U}$, Γ is a set of FDs and R satisfies all $\varphi \in \Gamma$.*

Delobel and Casey introduce in [12] a set of inference axioms and W. Armstrong in [3] proves that these axioms are correct and complete.

Proposition 2.3 (Armstrong's Axioms) *Let X, Y, Z be arbitrary subsets of the set $\mathcal{U}$ of attributes of a given relation R. We agree that XY stands for the union of X and Y. The following FD inference system is correct and complete:*

1. *If $Y \subseteq X$ then $X \mapsto Y$.* *(Axiom)*
2. *If $X \mapsto Y$ then $XZ \mapsto YZ$.* *(Augmentation Rule)*
3. *If $X \mapsto Y, Y \mapsto Z$ then $X \mapsto Z$.* *(Transitivity Rule)*

We say that the set of FDs Γ implies the FD $X \mapsto Y$ ($\Gamma \models X \mapsto Y$) if for all relation scheme $\mathcal{S} = (R, \Gamma)$, we have that R satisfies $X \mapsto Y$. The Armstrong's Axioms allow us to find all dependencies satisfied by a relational scheme $\mathcal{S}$, i.e. to find all the FDs $X \mapsto Y$ such as $\Gamma \models X \mapsto Y$.

This concept is named in the literature as Armstrong's relation and may be defined as the closure of a set the FDs in the following way:

Definition 2.4 (Closure of a FD set) *Let $\mathcal{S} = (R, \Gamma)$ be a relation scheme and $\Gamma \subseteq FD_{\mathcal{S}}$ and $X \subseteq \mathcal{U}$, we define the closure of the set of FDs Γ as*

$$\Gamma^+ = \{X \mapsto Y \in FD_{\mathcal{S}} \mid \Gamma \models X \mapsto Y\}$$

The implication problem to obtain $\Gamma \models X \mapsto Y$ can be fulfilled if $\{X \mapsto Y\} \subseteq \Gamma^+$. But, the method to calculate Γ^+ is a NP-algorithm. We need another way to attach the problem.

The algorithms to manipulate FDs that appear in the database literature utilize exhaustively the closure operator of a set of attributes: *Given X, a subset of attributes included in an attributes set $\mathcal{U}$, and $\Gamma \subseteq FD_{\mathcal{S}}$ its closure for Γ, denoted X^+, is a new set containing all the attributes deduced from X using Armstrong's Axioms* over Γ.

Definition 2.5 (Closure of an attribute set) *Let $\mathcal{S} = (R, \Gamma)$ be a relation scheme and $\Gamma \subseteq FD_{\mathcal{S}}$ and $X \subseteq \mathcal{U}$, we define the closure of an attribute set X as*

$$X^+ = \{A \in \mathcal{U} \mid X \mapsto A \in \Gamma^+\}$$

The following theorem, well established in the literature, reduce the implication problem to the closure of an attribute set problem. Since there exist several methods to compute X^+ in linear time, the implication problem may be solved with linear cost.

Theorem 2.6 $\Gamma \models X \mapsto Y$ *if* $Y \subseteq X^+$.

In computer science, the calculation of X^+ is profusely used in the database literature and is one of the key points in many problems: redundant dependency elimination, query optimization, the finding key problem, etc. Specifically, the most frequently used algorithms to transform relational databases in more efficient ones, use exhaustively the closure of a set of attributes (see[13, 22]).

Several algorithms are proposed in the database literature to obtain the closure of a set of attributes. The first linear algorithm is from Beery and Bernstein in [5]. We have implemented in C++, the most important algorithms that appear in the literature.

3 Classical closure algorithms

A non linear closure algorithm for a set of attributes appears in [20, 31] (see Algorithm 1). Consider $\mathcal{U}$ to be a set of attributes and Γ to be a set of FDs, then its complexity is $O(|\mathcal{U}||\Gamma|^2)$ in the worst case. A complexity study of this algorithm is available in [20].

Algorithm 1 ***Standard Closure***

```
Input:   $\mathcal{U}$, $\Gamma$, $X \subseteq \mathcal{U}$
Output:  $X^+$
Begin
    $X^+ = X$
    Repeat
        For each $A \mapsto B \in \Gamma$ do
            If $A \subseteq X^+$ and $B \not\subseteq X^+$ then
                $X^+ := X^+ \cup \{B\}$
            End if
        End for each
    Until no more attributes are added to $X^+$
    Return $X^+$
End
```

In the literature, there exist a set of works devoted to reduce the complexity of this algorithm, because an efficient execution of it, is essential to solve the implication problem.

In [5, 13, 26] several closures algorithms (algorithms 3, 4, and 5 in our empirical study), all of them with linear complexity, are presented. They use some data structures to reduce the cost of traversing the sets Γ (FDs) and $\mathcal{U}$ (attributes), decreasing the cost of the X^+ computation. The most common reduction strategies are the following: [1]

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

(b) To use an array indexed by the atomic attributes A_i that keeps 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 to the left-hand side for each FD that are not jet in the closure.

We remark that the first linear-time membership algorithm is presented in [5] by Beeri and Bernstein.

A detailed explanation of this algorithm appears in the PhD Thesis of S. Torgersen [30]. Torgersen cites the improvements presented for the first time in [5]: "The naive approach would be to find all dependencies with left-hand side containing attributes that are in X. Then X is augmented with the right-hand side attributes of these dependencies and this containment search is continued on the remaining dependencies until no left-hand side contains X. The results will be X^+."

The great efficiency improvement of these algorithms comes from the idea of adding the right hand side of each FD once we have checked that all their attributes are in the temporal closure. In this way, the algorithms traverse the set of FDs only once.

In the literature of AI and database, no closure algorithms use directly a formal base (Armstrong's Axioms or an FD logic). Nevertheless, the use of FDs in AI areas (data mining, rough set theories, knowledge representation, etc.) requires the development of automated deduction methods based on logic to manipulate FDs.

In the next section, we show a novel closure algorithm based in logic that computes efficiently the closure of a set of attributes.

4 The novel $\mathbf{SL}_{FD}$-closure algorithm

In [11] we have proposed a novel FD logic named *Substitution Logic for FDs* ($\mathbf{SL}_{FD}$) appropriated to develop automated deduction methods. It is guided by the idea of removing redundant attributes in an efficient way. This is one of the novelties of $\mathbf{SL}_{FD}$ logic because other well-known FD logic systems are guided by Armstrong Relations [3], more oriented to capture all the FDs that can be deduced from a given set of FDs.

In [11] there also are several examples that show how $\mathbf{SL}_{FD}$ directly removes redundancy in a set of FDs. In [25] we have developed a preprocessing pruning to remove redundancy in a FD set and we carry out an empirical study to prove the practical benefits of this approach. We remark that $\mathbf{SL}_{FD}$ is an efficient tool and we have shown the advantages of using a logic.

In [1] we prove that it is possible to use the paradigm of rewriting system to remove redundancy in FD sets and we use the Maude rewriting language to translate directly the rules of $\mathbf{SL}_{FD}$ and to use in an easy way these rules to remove redundancy automatically.

As a direct application of $\mathbf{SL}_{FD}$ in [24] we propose a framework to develop automated deduction methods for FDs. In this section, we show the $\mathbf{SL}_{FD}$-Closure algorithm based in this framework and we present an executable version of it (see algorithm 2).

The soundness and completeness of this algorithm are a direct consequence of the properties of the $\mathbf{SL}_{FD}$, presented in [11].

In the worst case, the *Repeat* loop (label *1* of the algorithm) is executed at most $|\ \mathcal{U}\ |$ times, since in every iteration at least one attribute is added to X_{new}. The *For* loop (label *2* of the algorithm)

[1] A deeper explanation of the classical algorithms is behind the scope of the paper and may be easily obtained using the references cited in the paper.

is executed at most $\mid \Gamma \mid$ times. Consequently, the complexity of the algorithm is $O(\mid \mathcal{U} \mid\mid \Gamma \mid)$. 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.

Algorithm 2 ***Linear*** $\mathbf{SL}_{FD}$***-Closure.***

```
Input
 $\mathcal{U}$: a set of attributes,
 $\Gamma$: a set of FDs,
 $X$: a subset of attributes
Output
 $X^+$: the closure of $X$
Begin
      $X_{new}$ := $X$
      $X_{old}$ := $X$
[1]   Repeat     [1]
        $X_{old}$ := $X_{new}$
[2]     For each $A \mapsto B \in \Gamma$ do [2]
          If $A \subseteq X_{new}$ then
            $X_{new} := X_{new} \cup B$                  (Case I)
            $\Gamma = \Gamma - \{A \mapsto B\}$
          elsif $B \subseteq X_{new}$ then
            $\Gamma = \Gamma - \{A \mapsto B\}$                (Case II)
          else
            $\Gamma = \Gamma - \{A \mapsto B\}$               (Case III)
            $\Gamma = \Gamma \cup \{A - X_{new} \mapsto B - X_{new}\}$ (Case III)
          End if
        End for each
      Until (($X_{new} = X_{old}$) or ($|\Gamma| = 0$))
      Return $X^+ = X_{new}$
End
```

We remark that in the application of the *Case III*, if $A \cup X_{new} = \varnothing$ and $B \cup X_{new} = \varnothing$ then it renders the same Γ. In figure 1, this situation is labelled as *none*.

Example 1 *Let Γ be the following set of dependencies:$\{ag \mapsto h, h \mapsto cde, e \mapsto kl, ke \mapsto m, fc \mapsto j, d \mapsto bei, j \mapsto c, im \mapsto g\}$ and let $X = ad$ be a set of attributes.*

We apply the closure algorithm that renders $\{X_{new} : adbeiklmghc, f \mapsto j\}$ and, therefore, the closure of (ad) is $(ad)^+ = adbeiklmghc$.

Figure 1 shows step by step the application of the $\mathbf{SL}_{FD}$ *closure algorithm. By rows we depicted the iterations of the* Repeat *loop. We also label each FD with the applied Case (I,II,III or none). We put the symbol $\times$, if the FD is removed from Γ. Moreover, we illustrate the increase of the expression X_{new} when Case I is applied.*

In the following section, we compare our method to the other algorithms. We remark that we don't use any data structure to summarize the information of the FDs . We only use the rules of the $\mathbf{SL}_{FD}$ logic and we could improve the behavior if we use the reduction strategies showed in the section 3.

5 The design of the empirical study

In this paper, we summarize the background about closure algorithms and a new closure algorithm with a formal base, the $\mathbf{SL}_{FD}$ logic, is showed. The complexity in the worst case is the same as the classical linear closure algorithms. Furthermore, in this work we have carried out an empirical study about the execution time [2] of closure algorithms. To do so we have implemented them using C^{++} in a Pentium 4 (1500 MHz, 512 MB of RAM, Windows 2000).

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 *ran3*. This algorithm is shown in [27] following a Knuth's suggestion and it is a portable routine based on a subtractive method. We labelled the algorithms that we have developed in C++:

- Algorithm 2: Linear $\mathbf{SL}_{FD}$-Closure.
- Algorithm 3: Beeri Linear Closure.
- Algorithm 4: Diederich Linear Closure.
- Algorithm 5: Paredaens Linear Closure.

[2] In [33] the author remark that 'the execution time are mostly relevant for comparing algorithms'.

Γ		$ag\mapsto h$	$h\mapsto cde$	$e\mapsto kl$	$ke\mapsto m$	$fc\mapsto j$	$d\mapsto bei$	$j\mapsto c$	$im\mapsto g$
X_{new} :	ad						$adbei$		
$Case$:		(III)	(III)	none	none	none	(I)	none	(III)
Γ'		$g\mapsto h$	$h\mapsto ce$	$e\mapsto kl$	$ke\mapsto m$	$fc\mapsto j$	$\times$	$j\mapsto c$	$m\mapsto g$
X_{new} :	$adbei$			$adbeikl$	$adbeiklm$				$adbeiklmg$
$Case$:		(III)	(III)	(I)	(I)	none		$none$	(I)
Γ'		$g\mapsto h$	$h\mapsto c$	$\times$	$\times$	$fc\mapsto j$		$j\mapsto c$	$\times$
X_{new} :	$adbeiklmg$	$adbeiklmgh$	$adbeiklmghc$						
$Case$:		(I)	(I)			(III)		(II)	
Γ'		$\times$	$\times$			$f\mapsto j$		$\times$	

Figure 1: Example 1

In order to have another criterion measure, we use another parameter defined in [4, 33]. The authors define the *size* of a FD set as follows:

Definition 5.1 *Let be* $\Gamma = \{X_1\mapsto Y_1, \ldots, X_n\mapsto Y_n\}$ *a set of FDs, we define the size of* Γ *as* $\parallel \Gamma \parallel = \sum_{i=1}^{n}, (\mid X_i \mid + \mid Y_i \mid)$ [3].

We randomly generate a set of attributes $\mathcal{U}$, a set of FDs Γ, and a subset of attributes $X \subseteq \mathcal{U}$ and we apply the 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 closure algorithms in the order detailed previously.

In this study, we have executed the closure algorithms 1.817 times. We have generated FD sets with 25, 50,75,100,125,150,175 and 200 FDs. And for each FD set with N FDs, we have repeated the execution of the algorithms varying the maximum number of attributes in the left-hand side and in the right-hand side, from 1 to 300. The range in the size of the FD sets has varied from 50 to 61770 attributes.

We have calculated the average of the 1.817 executions and the results are in the following table:

Method	average	median
Diederich Closure	4.593,48	1.001,50
Beeri Closure	7.013,56	4.139,33
Paredaens Closure	5.863,35	1.643,00
$\mathbf{SL}_{FD}$-Closure	1.262,41	230,00

Table 1

$\mathbf{SL}_{FD}$-Closure has had a significatively better behavior than other linear closures. $\mathbf{SL}_{FD}$-Closure has not only spent less time than the other algorithms but also has a narrow 95 % confidence interval on the mean that is strictly under the confidence intervals of the others. It can be seen in the following figure.

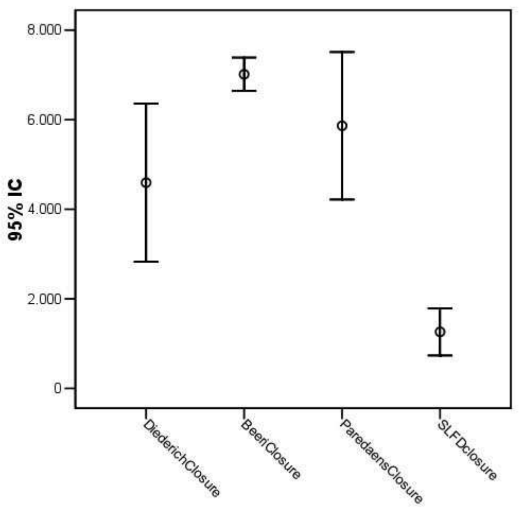

Figure 2. Empirical study

[3]Where $\mid X \mid$ is the cardinality of the set X.

6 Conclusions and future works

In this paper, most important closure algorithms have been studied. We have selected for comparing some of them that has the lower complexity, that is linear complexity.

Algorithm $\mathbf{SL}_{FD}$-Closure is described as a direct application of the rules of the $\mathbf{SL}_{FD}$ logic. Its complexity in the worst case is also linear.

An empirical study has been carried on. This study has compared the time spent for each closure algorithm in 1.817 FD sets with different cardinality (from 25 to 200 FDs) and different size (from 50 to 61770 attributes).

$\mathbf{SL}_{FD}$-Closure Algorithm has spent significatively less time than the other linear closure algorithms.

Actually, we work in the use the of $\mathbf{SL}_{FD}$-Closure algorithm to improve the pre-processing transformation algorithm that we use in [25] to obtain the canonical closure of a set of FDs.

A future work will be the study of the behavior of closure algorithms with respect to different patterns of FD sets.

Also, we are applying AI techniques in database. We have developed a CASE tool to design collaborative database. In a collaborative environment, it is necessary to make the integration of the different users schemes and to summarize all the information in a unique and unified system. The goal is *to integrate all the information in a global model without the existence of redundances and inconsistences and to answer the users global queries searching the data in the heterogeneous and probably disperse data sources.*

We propose the use of the $\mathbf{SL}_{FD}$ logic to carry out the integration process using an intelligent deduction method. And we will use $\mathbf{SL}_{FD}$-Closure to optimize query in the unified data model.

7 Annex

In all algorithms the input is $\mathcal{U}$ a set of attributes, Γ a set of FDs and $X \subseteq \mathcal{U}$ and subset of attributes. The output is the closure of X denoted by X^+.

Algorithm 3 *Beeri Linear Closure.*

```
Input:   $\mathcal{U}$, $\Gamma$, $X \subseteq \mathcal{U}$
Output: $X^+$
Begin
 For i=1 to n do
   $AttrList[i] = NIL$
   For j=1 to m do
     If $A_i \in FD_j.lhs$ then
       $AttrList[i] = AttrList[i] \cup \{j\}$
     End if
   End for
 End for
 For j=1 to m do
   $Counter[j] = \| FD_j.lhs \|$ then
 End for
 $X^+$, $AddedAttr = X$
 While ($AddedAttr \neq \varnothing$) do
   $AddedAttr = AddedAttr - A_i$
   For each $FD_j \in AttrList[i]$ do
     $Counter[j] = Counter[j] - 1$
     If $Counter[j] = 0$ then
     $AddedAttr = AddedAttr \cup (FD_j.rhs - X^+)$
     $X^+ = X^+ \cup FD_j.rhs$
     End if
   End for each
 End while
 Return $X^+$
```

This algorithm use the following strategies to improve the complexity:

- The authors use $Counter[j]$ to store is the number of attributes in $FD_j.lhs$ not in X^+.
- The authors use *AddedAttr* to store a subset of Closure where each element is added exactly once.
- The authors use *AttrList*$[i]$ to store a list of FDs whose left sides contain A_i.

Algorithm 4 *Diederich Linear Closure*

```
Input:   $\mathcal{U}$, $\Gamma$, $X \subseteq \mathcal{U}$
Output: $X^+$
Begin
   $X^+ := X$
   $UPDATE := X$
   For each $A \mapsto B$ do
      $COUNT[A \mapsto B] := | A |$
   End for each
   For each attribute $A_i$
      Construct $LIST[A_i]$
   End for each
   While $UPDATE$ is not empty
      select and remove an attribute
      $A_i$ from $UPDATE$
      For each $A \mapsto B \in LIST[A_i]$ do
          decrement $COUNT[A \mapsto B]$
          If $COUNT[A \mapsto B] = 0$ then
            add $B$ to $UPDATE$ and $X^+$
            if $B$ is not already in $X^+$
          End if
        End for each
    End while
    Return $X^+$
```

The great efficiency improvement of this algorithm comes from the idea of adding the right hand side of each FD once we have checked that all their attributes are in the temporal closure. In this way, the algorithm traverses the set of FD only once.

The Algorithm 4 use the following strategies to improve the complexity of the closure algorithm:

- The authors use $COUNT[A \mapsto B]$ to store the number of attributes in the left-hand side.
- The authors use $LIST[A_i]$ to store a list of pointers to dependencies in which $A_i \in A$.

Algorithm 5 ***Paredaens Linear Closure.***

```
Input:  $\mathcal{U}, \Gamma, X \subseteq \mathcal{U}$
Output: $X^+$
Begin
    $X^+ = \varnothing$
    $XWAIT = X$
    For each $X \mapsto Y \in \Gamma$ do
        $NOTIN(X \mapsto Y) = | X |$
        If $X = \varnothing$ then
            $XWAIT = XWAIY \cup Y$
        End if
        For each $A \in X$ do
            $INLFD[A] = INLFD[A] \cup \{X \mapsto Y\}$
        End for each
    End for each
    While $XWAIT \neq \varnothing$
        For each $A_k \in XWAIT$ do
            $XWAIT = XWAIT - \{A_k\}$
            $X^+ = X^+ \cup \{A_k\}$
            For each $X \mapsto Y \in INLFD(A_k)$ do
                $NOTIN(X \mapsto Y) = NOTIN(X \mapsto Y) - 1$
                If $NOTIN(X \mapsto Y) = 0$ then
                        $XWAIT = XWAIT \cup \{Y - X^+\}$
                End if
            End for each
        End for each
    End while
    Return $X^+$
```

In [26] Paredaens et. al show that the complexity of these algorithm is $O(| \mathcal{U} || \Gamma |)$. Also, they mention that "in the literature $O(| \mathcal{U} || \Gamma |)$ 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".

Paredaens et al. [26] apply some of the previous strategies to improve the complexity of the closure algorithm:

- The authors use $NOTIN[X \mapsto Y]$ to store the number of attributes in the left-hand side.
- The authors use $INLFD[A]$ a list of pointers to dependencies in which $A_i \in A$.

References

[1] Gabriel Aguilera, Pablo Cordero, Manuel Enciso, Angel Mora, and I. P. de Guzmán. A non-explosive treatment of Functional dependencies using rewriting logic. *LNAI 3171, Springer-Verlag (SBIA 2004)*, pages 31–40, 2004.

[2] Marcelo Arenas and Leonid Libkin. An information-theoretic approach to normal forms for relational and xml data. *In PODS,San Diego, CA, USA*, 2003.

[3] William W. Armstrong. Dependency structures of data base relationships. *Proc. IFIP Congress. North Holland, Amsterdam*, pages 580–583, 1974.

[4] Paolo Atzeni and Valeria De Antonellis. Relational Database Theory. *Ed. The Benjamin/Cummings Publishing Company Inc.*, 1993.

[5] C. Beeri and P. A. Bernstein. Computational Problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems*, 4 (1):30–59, 1979.

[6] D. A. Bell. From data properties to evidence. *IEEE Transactions on Knowledge and Data Engireering*, 5 (6):965–968, 1993.

[7] Elisa Bertino, Barbara Catania, and Gian Piero Zarri. Intelligent database systems. *Ed. ACM Press. Addison-Wesley*, 2001.

[8] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for xml. *Draft manuscript*, 2000.

[9] Nathalie Caspard and Bernard Monjardet. The lattices of closure systems, closure operators, and implicational systems on a finite set: a survey. *Discrete Applied Mathematics*, 127 (2):241–269, 2003.

[10] Edgar F. Codd. A relational model of data for large share data banks. *Comm. ACM*, 13 (6):377–387, 1970.

[11] Pablo Cordero, Manuel Enciso, I. P. de Guzmán, and Angel Mora. SLFD logic: Elimination of data redundancy in knowledge representation. *LNAI 2527, Springer-Verlag (IBERAMIA 2002)*, pages 141–150, 2002.

[12] C. Delobel and R.G. Casey. Decomposition of a data base and the theory of boolean switching functions. *IBM J. of Research and Development*, 17 (5), 1973.

[13] Jim Diederich and Jack Milton. New methods and fast algorithms for database normalization. *ACM Transactions on Database Systems*, 13 (3):339–365, 1988.

[14] J. W. Guan and D. A. Bell. Rough computational methods for information systems. *Artificial Intelligence*, 105 1 (2):77–103, 1998.

[15] Erika Hajnicz. Time structures. Formal description and algorithmic representation. *LNAI 1047, Springer-Verlag*, 1996.

[16] S. Hartmann and S. Link. The implication problem of functional dependencies in complex-value databases. *Electronic Notes in Theoretical Computer Science*, 123:125–137, 2005.

[17] Toshihide Ibaraki, Alexander Kogan, and Kazuhisa Makino. Functional dependencies in Horn Theories. *Artificial Intelligence*, 108 1-2:1–30, 1999.

[18] M. Kirchberg and S. Link. On the implication problem for functional dependencies in the higher-order entity-relationship model. *CRPITS'17: Proceedings of the Fourteenth Australasian database conference on Database technologies 2003*, pages 115–124, 2003.

[19] M. L. Lee, T. W. Ling, and W. L. Low. Designing functional dependencies for xml. *LNCS, Springer-Verlag*, 2287:124–141, 2002.

[20] David Maier. The theory of relational databases. *Computer Science Press*, 1983.

[21] H. Mannila. Methods and problems in data mining. *Proceedings of International Conference on Database Theory. Afrati, Kolaitis (ed.)*, 17 (2), 1997.

[22] H. Mannila and K. Raiha. Design of relational databases. *Ed. AddisonWesley, Reading, MA*, 1992.

[23] H. Mannila and K. Raiha. Algorithms for inferring functional dependencies from relations. *Data and Knowledge Engineering*, 12 (1):83–99, 1994.

[24] Angel Mora, Manuel Enciso, Pablo Cordero, Gabriel Aguilera, and I.P.de Guzmán. Closure via functional dependence simplification. *Submitted to Acta Informatica*, 2005.

[25] Angel Mora, Manuel Enciso, Pablo Cordero, and Inmaculada P. de Guzmán. An efficient preprocessing transformation for functional dependencies sets based on the substitution paradigm. *LNAI, Springer-Verlag (CAEPIA 2003)*, 3040, 2004.

[26] Jan Paredaens, Paul De Bra, Marc Gyssens, and Dirk Van Van Gucht. The structure of the relational database model. *EATCS Monographs on Theoretical Computer Science. Ed. Springer-Verlag New York, Inc.*, 1989.

[27] William H. Press and et al. Numerical recipes in C. The art of Scientific Computing. ed. Cambridge University Press. 1999.

[28] David Robertson and Jaum Agustí. Lightweight uses of logic in conceptual modelling. *Software Blueprints. ACM Press. Ed. Addison Wesley*, 1999.

[29] Bernhard Thalheim. An overview on semantical constraints for database models. *6th International Conference, Intellectual Systems and Computer Science. Moscow, Russia.*, 1996.

[30] Solveig Torgersen. Automatic design of relational databases. *Ph. D. Thesis. TR 89-1038. Cornell University, Ithaca*, 1989.

[31] Jeffrey D. Ullman. Principles of database systems. *Computer Science Press*, 1982.

[32] Lin Lin Wang. Thorough investigations into : An improved algorithm based on subset closures for synthesizing a relational database cheme. *IEEE Transactions On Software Engineering*, 22 (4):271–274, 1996.

[33] Marcel Wild. Computations with finite closure systems and implications. *Computing and Combinatorics. LNCS 959. Springer, Berlin-Heidelberg*, pages 111–120, 1995.