Non first normal form relations:An algebra allowing data restructuring

Serge Abiteboul, Nicole Bidoit

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. : (1) 39 63 55 11

Rapports de Recherche

N° 347

NON FIRST NORMAL FORM RELATIONS : AN ALGEBRA ALLOWING DATA RESTRUCTURING

Serge ABITEBOUL
Nicole BIDOIT

Novembre 1984

NON FIRST NORMAL FORM RELATIONS :

AN ALGEBRA ALLOWING DATA RESTRUCTURING

Serge ABITEBOUL
and
Nicole BIDOIT (*)

Institut National de Recherche en Informatique et Automatique
78153 Le Chesnay Cedex, France.

December, 1985.

(*) Current adress : Laboratoire de Recherche en Informatique, Universite Paris-Sud, 91405 ORSAY, France.

Abstract

A database model based on non first normal form relations is presented. A key feature of the model is an algebraic query language allowing data restructuring. A natural connection between instances in this model and, in the relational model under the Universal Relation Scheme assumption is investigated.

Résumé

Un modèle de base de données utilisant des relations non sous première forme normale est présenté. Un aspect essentiel du modèle est l'existence d'un langage algébrique de requête autorisant la restructuration des données. On présentera aussi un lien natural entre les instances de ce modèle, et les instances relationnelles satisfaisant le Postulat du Schéma Universel.

INTRODUCTION

Several investigators have stressed that the first normal form (1NF) condition [Co] is not convenient for handling a variety of database applications [Mak, K, Mac]. The first purpose of this paper is to present a database model, namely, the Verso model, where data is organized in non 1NF relations. The values for some attributes in a Verso instance are atomic whereas the values for other attributes are simpler Verso instances. As we shall see, this recursive definition of the data structure induces a hierarchical organization of the data. It should be noted that the notion of hierarchical data organization has been captured in some form by at least two other models [IMS, HY]. The advantage of our approach is that, by using relation as underlying structure, we are able to preserve some of the positive features of the relational model, for instance a simple algebraic query language.

As mentioned earlier, the first major theme of this paper is to formally present the data structure and operations in Verso. In a Verso schema, some dependencies (very similar to Delobel's Generalized Hierarchical Dependencies [D]) are implicitely specified. Therefore, some semantic connections among the attributes are implied by the choice of a Verso schema. Furthermore, the operations that we propose on Verso instances take advantage of these semantic connections. In particular, some queries which would typically require joins in the pure relational model can be expressed by a selection in the Verso model removing the need for the user to specify access paths.

The second major theme of the paper is the investigation of some key issues raised by this data organization. First, data restructuring is studied via the notions of schema equivalence and dominance. Necessary and sufficient conditions for schema equivalence and dominance are exhibited based on some elementary schema transformations.[#] Also, a natural connection between Verso instances and relational database instances satisfying the Universal Relation Schema Assumption [FMU, MW] is investigated. This allows us to (1) give an interpretation of the operations in terms of (pure) relational operations, and (2)

[#] The notion of data restructuring is studied in depth by Hull and Yapp [HY] for a very large class of hierarchical data structures. By opting for a more restricted model, we are capable here to develop an algebra which incorporates restructuring.

measure the power of the Verso operations by proving that they are "complete".

Non 1NF relations have recently attracted a lot of attention. A model is introduced in [Mac] which describes some data structures very similar to the ones presented here. However, the access language exhibited there is quite weak, and in particular does not allow data restructuring. An algebra for non 1NF relations of non necessarily hierarchical structure is also proposed in [SS]. Other aspects of non normalized relations have been studied in [AMM, FK, FT, KTT, JS, SP].

1. PRELIMINARIES

In the following, we assume that the reader is familiar with the relational model. In this section, we briefly review some well-known concepts, and present the notation used throughout the paper.

We assume the existence of an infinite set U of **attributes**, and for each A in U, of a set of **values** called the **domain** of A and denoted dom(A). A **relational schema** is a finite set of attributes. Let V be a relational schema. A **tuple** v over V is a mapping from V into $\bigcup_{A \text{ in } V} \mathrm{dom}(A)$ such that v(A) is in dom(A) for each A in U.

A **(1NF) relation** over V is a finite set of tuples over V. The set of tuples over V is denoted tup(V), and the set of relations rel(V). The relational operations of **union**, **intersection**, **difference**, **join**, **projection**, and **selection** are respectively denoted $\cup$, $\cap$, $-$, $*$, π, and $\mathrm{select}_{[C]}$ (where C is an elementary condition of the form $A < a$, $A \leq a$, $A = a$, $A \geq a$, $A > a$, for some A in U and a in dom(A)).

A **relational database schema** is a finite set of relational schemas. A **relational (database) instance** r of some relational database schema R is a mapping from R such that, for each X in R, r(X) is in rel(X). A relational instance satisfies the **Universal Relation Schema Assumption (URSA)** iff $r(X) \supseteq \pi_X(r(Y))$ for each X, Y in R and $X \subseteq Y$.

In the paper, we also consider finite **strings** of attributes. Let $A_1 \cdots A_n$ be a finite string of attributes. An **ordered tuple** x over $A_1 \cdots A_n$ is an element of the cartesian product $dom(A_1) \times \cdots \times dom(A_n)$. The set of ordered tuples over some string X is denoted Otup(X).

For each string X of attributes, the corresponding set of attributes, i.e., $\{A \mid A \text{ in } X\}$ is denoted set(X). For each ordered tuple x over X, the corresponding tuple over set(X) is denoted map(x). Note that map(x) is a mapping.

In general, A, B, ... denote attributes, a, b, ... values, V, W, X, Y, ... relational schemas (or finite strings of attributes), v, w, x, y, ... (ordered) tuples, R, S, ... relational database schemas, and r, s, ... relational database instances. We also use the classical convention of writing XY for the union of two sets X and Y of attributes, or for the concatenation of two strings X and Y of attributes.

2. THE MODEL

In this section, we present the data structure of the Verso model (namely, the Verso instance) using the auxiliary concept of format. We then introduce five unary operations (extension, projection, selection, restriction, and renaming), and five binary ones (union, intersection, difference, join, and cartesian product). As we shall see, Verso instances offer a generalization of relational instances. Furthermore, some of these operations generalize relational operations.

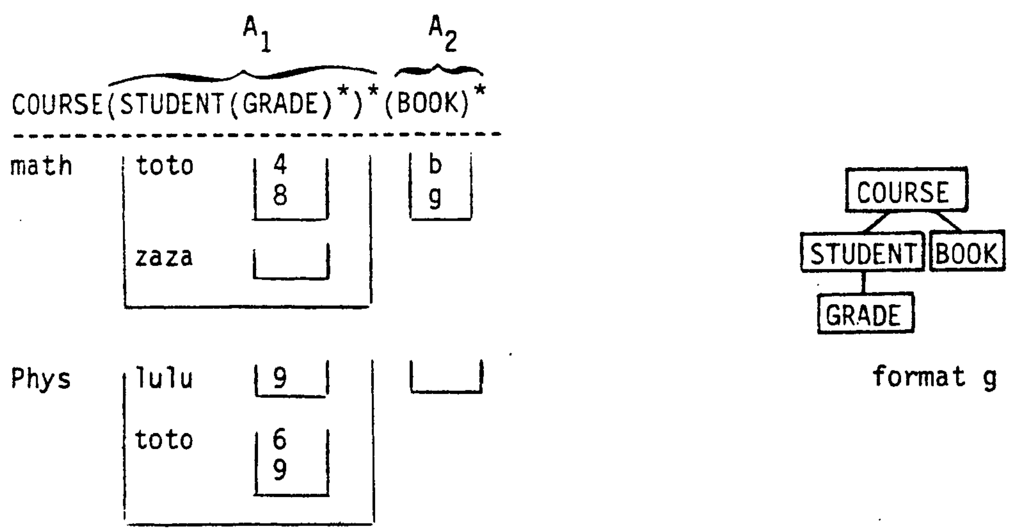
Let us first consider an example. A department consists of a set of COURSEs, the BOOKs for each course, the STUDENTs in the course, and their GRADEs. We can represent an instance of a department like in Figure 1(a). Intuitively, a department can be considered as a relation over three attributes, say COURSE, A_1 and A_2. The values in dom(COURSE) are atomic whereas the values in $dom(A_1)$ and $dom(A_2)$ are simpler Verso instances. Let us make two remarks. The first one is that, in the example, there is no book required in the physics course. (Thus, null values of the type "does not exist" can be represented in a Verso instance). The second remark is that an implicit connection is assumed between the attributes STUDENT and BOOK through the attribute COURSE.

In order to formalize the notion of Verso instance, we need the auxiliary concept of format. Intuitively, a format specifies the underlying structure of a Verso instance.

Definition : A **format** is recursively defined by:

(1) Let X be a finite string of attributes with no repeated attribute, then X is a **(flat) format** over the set of attributes occuring in X, i.e. set(X), and

(2) Let X be a non empty finite string of attributes with no repeated attribute, and $f_1, \ldots, f_n$ some formats over $Y_1, \ldots, Y_n$, resp., such that the sets set(X), $Y_1, \ldots, Y_n$ are pairwise disjoint, then the string $X\,(f_1)^* \cdots (f_n)^*$ is a **format** over the set $set(X)Y_1 \cdots Y_n$.



(a) (b)

- *Figure 1* - Format and Verso Instance.

For instance, $f =$ COURSE STUDENT GRADE is a flat format over {COURSE, STUDENT, GRADE}, and $g =$ COURSE(STUDENT(GRADE)*)*(BOOK)* is a format over {COURSE, STUDENT, GRADE, BOOK}.

In the following, Λ denotes the empty string. (By definition, Λ is a format.) Also, if $f \equiv X\,(f_1)^* \ldots (f_n)^*$ is a format, and $f_i \equiv \Lambda$ for some i, then we identify f and $X\,(f_1)^* \ldots (f_{i-1})^* (f_{i+1})^* \ldots (f_n)^*$.

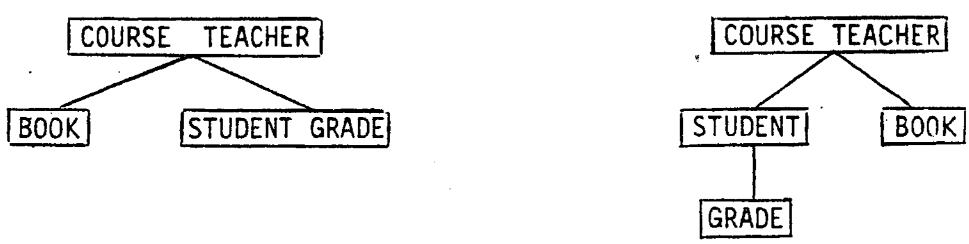
In the following, we shall use a directed tree representation for formats. The tree representation of the format g is given in Figure 1(b). Other examples of formats are given in Figure 2. Let $f \equiv X\,(f_1)^* \ldots (f_n)^*$ be a format. Then X is called the **root** of f, and each f_i a **branch**.

We are now able to formally define the Verso instances.

Definition: Let f be a format. The set of all **(Verso) instances** over f, denoted inst(f), is recursively defined by:

(i) if $f \equiv X$, and X is non empty, then I is in inst(f) iff I is a finite subset of Otup(X), and

(ii) if $f \equiv X\,(f_1)^* \ldots (f_n)^*, f_1, \ldots, f_n$ non empty, then I is in inst(f) iff
  (a) I is a finite subset of $\mathrm{Otup}(X) \times \mathrm{inst}(f_1) \times \cdots \times \mathrm{inst}(f_n)$, and
  (b) if $<u, I_1, \ldots, I_n>$ and $<u, J_1, \ldots, J_n>$ are in I for some u,



COURSE TEACHER(BOOK)*(STUDENT GRADE)*

COURSE TEACHER(STUDENT(GRADE)*)*(BOOK)*

- *Figure 2* - Tree Representation of Formats.

$I_1, \ldots, I_n, J_1, \ldots, J_n$ then $J_i = I_i$ for each i in [1..n].

In the previous definition, we assume for (ii) that the formats $f_1, \ldots, f_n$ are non empty. Now, if $f \equiv X\,(f_1)^*\ldots(f_n)^*$ with $f_i \equiv \Lambda$ for some i, and $f_j \not\equiv \Lambda$ for $j \neq i$, then by convention, we identify f with $g \equiv X\,(f_1)^*\ldots(f_{i-1})^*(f_{i+1})^*\ldots(f_n)^*$, and the set of all instances over f is obtained from the previous definition by: inst(f) = inst(g).

Intuitively, the (i) condition states that I is atomic over the attributes in X, and not atomic over the "attributes" $f_1, \ldots, f_n$. The (ii) condition forces X to be a key. It is clear that the mathematical notation for Verso instances is cumbersome and not really readable. Therefore, in the following, instances will be represented using the "bucket" technique of [P] (See Figure 1(a)).

In the relational model, a database schema consists of several relational schemas. Similarly, we have :

Definition: A **Verso database schema** Ω is a finite set of formats. **A Verso database instance** σ of the schema Ω is a mapping from Ω to $\bigcup_{f \text{ in } \Omega} \text{inst}(f)$ such that $\sigma(f)$ is an instance over f for each f in Ω.

We now introduce an inclusion relation on Verso instances. Intuitively, an instance over some schema f is included in another instance over the same format f iff all the information contained in the first instance is also contained in the second one. Formally, we have:.

Definition: Let f be a format. Let I and J be two instances over f. Then I is **included in** J (or J **contains** I), denoted $I \leq J$, iff :

(i) if $f \equiv X$, X non empty, then $I \subseteq J$, and

(ii) if $f \equiv X\,(f_1)^* \cdots (f_n)^*, f_1, \ldots, f_n$ non empty, then :

$\forall$ $\langle uI_1 \cdots I_n \rangle$ in I, $\exists \langle uJ_1 \cdots J_n \rangle$ in J such that $I_i \leq J_i$ for each i in [1..n].

We shall use this inclusion relation and set operators to present the operations on Verso instances. We first present the unary operations on Verso instances. To do that, we need the auxiliary concept of subformat. Intuitively, g is a subformat of f if the tree representation of g can be obtained by pruning some terminal subtrees of the tree representation of f. Formally,

Definition: Let f be a format. Then a **subformat** g of f is recursively defined by:

(i) For each f, Λ is a **subformat** of f,

(ii) If $f \equiv X\,(f_1)^* ... (f_n)^*$, and $g_1, \ldots, g_n$ are respectively subformats of $f_1, \ldots, f_n$, then $X\,(g_1)^* ... (g_n)^*$ is a **subformat** of f.

Let f and g be two formats such that g is a subformat of f. Then, intuitively, it is possible to represent the information content of an instance over g by an instance over f. Indeed, the **extension** of an instance J over g **to** f, denoted J^f, is simply obtained by "padding" at each level with empty instances. We do not formally define the extension operation but illustrate the concepts of subformat and extension by the following example.

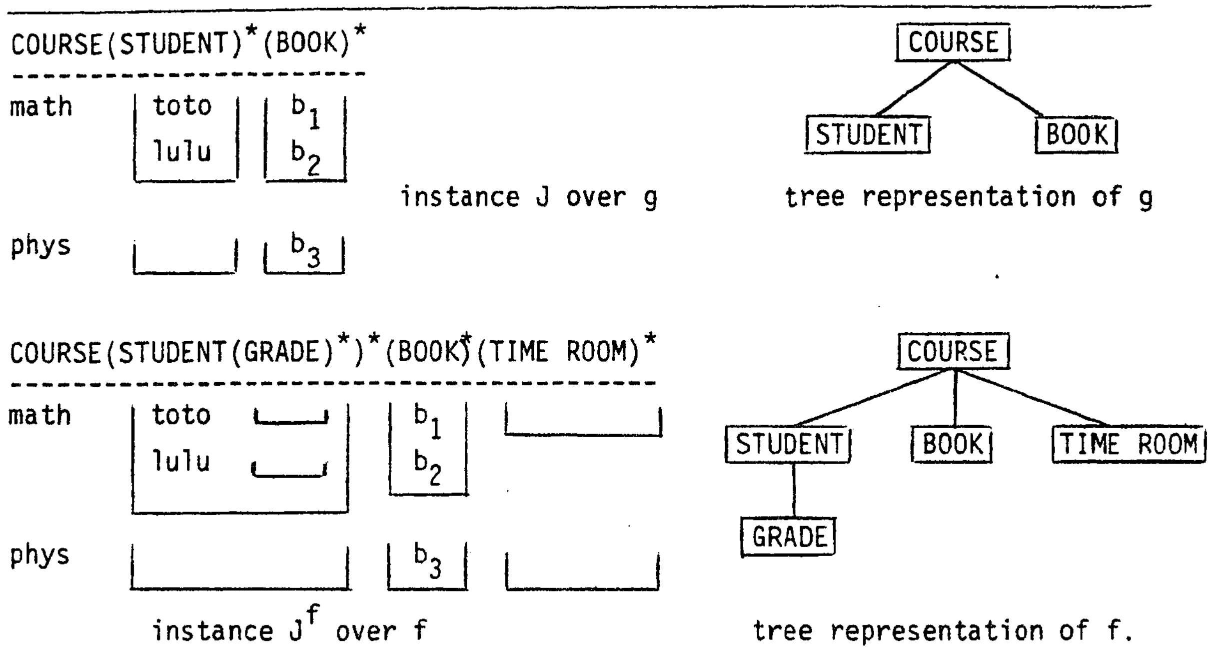
Example 2.1: The format g = COURSE(STUDENT)*(BOOK)* is a subformat of the format f = COURSE(STUDENT(GRADE)*)*(BOOK)*(TIME ROOM)*. The directed trees associated with f and g are represented in Figure 3, together with an instance J over g, and its extension J^f over f.

Note that in particular, each format f is a subformat of itself.

We now present the projection. Let I be an instance over f, and g a subformat of f, then the result of the projection of I over g is simply obtained by removing all the subinstances in I corresponding to subtrees of f which are projected out.

We propose two equivalent definitions of projection. (The proof of their equivalence is straightforward, and therefore omitted.) The first one uses the extension operator, and the inclusion relation on instances.

Definition: Let f and g be two formats such that g is a subformat of f, and $g \not\equiv \Lambda$. Let I be an instance over f. Then the **projection** of I **over** g, denoted I[g], is the greatest instance over g whose extension to f is included in I.



- *Figure 3* - Subformat and Extension.

In a constructive and equivalent way, we have:

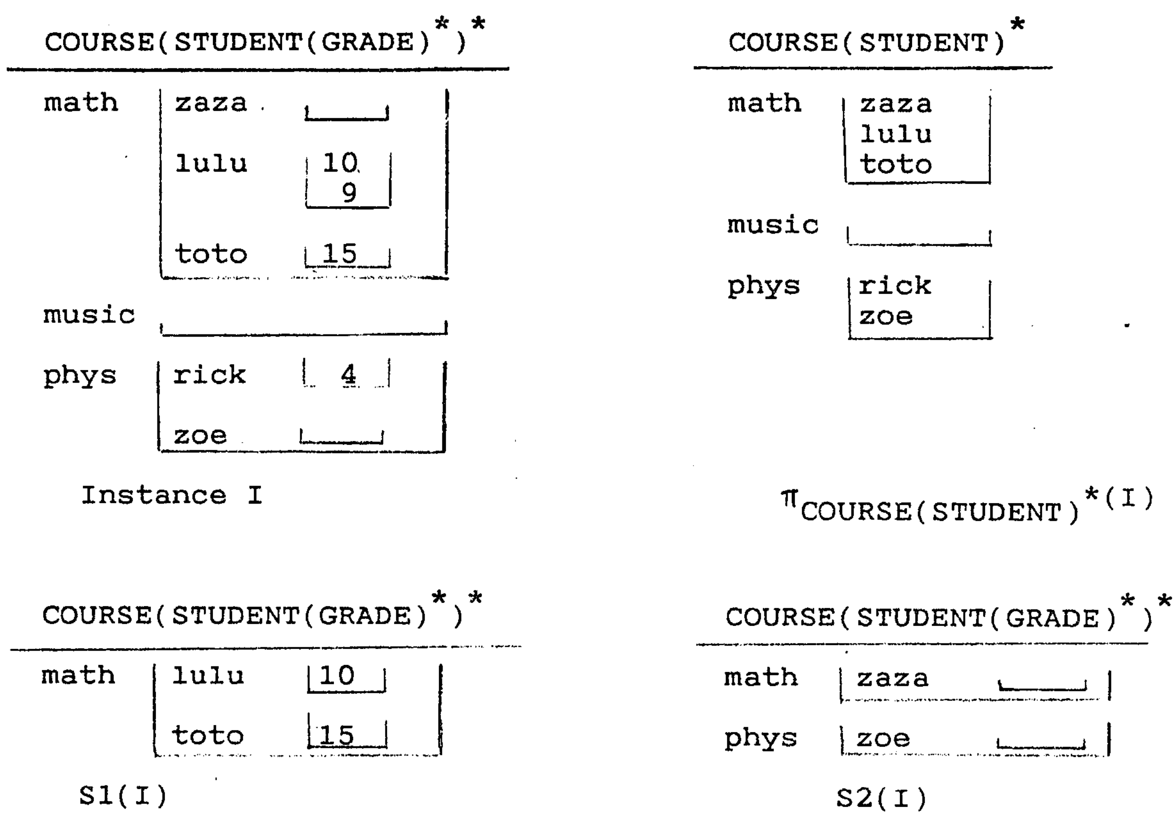
Definition: Let $f \equiv X\,(f_1)^* \ldots (f_n)^*$, $f_1, \ldots, f_n$ non empty, and $g \equiv X(g_1)^* \ldots (g_m)^*$ be two formats such that, for each j in [1..m], g_j is a non empty subformat of f_i for some i in [1..n]. Let I be an instance over f. Then the **projection** of I **over** g, denoted I[g], is recursively defined by:

$$I[g] = \left\{ <uJ_1 \cdots J_m> \;\middle|\; \begin{array}{l} \exists <uI_1 \cdots I_n> \in I, \text{ such that} \\ \forall\, j \text{ in } [1..m],\ J_j = I_i[g_j] \text{ where } g_j \text{ is a subformat of } f_i \end{array} \right\}.$$

An example of projection can be found in Figure 4.

Note that the projection as presented above does not generalize the relational projection. Indeed, for a flat format X, the only projection which can be performed is the projection over X, i.e., the identity mapping. However, it is shown in Section 5 that arbitrary projections can be performed using restructuring (presented in Section 4), and projection as defined here.

The third unary operation is the (Verso-)selection. This operation is more intricate than the relational selection since it takes advantage of the richer structure of Verso instances. In this section, we introduce a simple version of the selection. (A more powerful selection will be presented in Section 5.) In order to do this, we need the auxiliary concept of a condition on a sequence of



- *Figure 4* - Projection and Selection.

attributes.

Definition: Let X be a sequence of attributes. Then the **conditions** on X are obtained in the following way:

(1) each elementary condition on A for some A in X is a **condition** on X, and

(2) if C_1 and C_2 are conditions on X, then $(C_1 \wedge C_2)$, $(C_1 \vee C_2)$, and $(\neg C_1)$ are **conditions** on X.

The notion of **satisfaction** of a condition by an ordered tuple is defined in the straightforward way. Let C be a condition on X, and x an ordered tuple over X. Then x satisfies C is denoted $x \models C$.

We now define (the simple version of) the selection.

Definition: Let $f \equiv X\,(f_1)^* \ldots (f_n)^*$ be a format for some $n \geq 0$, $f_1, \ldots, f_n$ non empty, and I an instance over f. Then a **(Verso-)selection** S over f is an expression of the form: $S \equiv X : C\,(\,e_1(S_1)\,, \ldots,\, e_n(S_n)\,)$ where :

(a) C is a condition on X,

(b) for each i in [1..n], S_i is a selection over f_i, and

(c) for each i in [1..n], e_i is a symbol in $\{\exists, \nexists, ?\}$ ($\exists$ is read "exists", $\nexists$ "does not exist", and ? "does not care").

A selection defines an operation in the following way :

Definition: Let $f \equiv X\,(f_1)^* \ldots (f_n)^*$ be a format for some $n \geq 0$, with $f_1, \ldots, f_n$ non empty, and I an instance over f. Let $S \equiv X : C\,(\,e_1(S_1)\,, \ldots,\, e_n(S_n)\,)$ be a selection over f. Then the **result** of S **applied** to I, denoted S(I), is the instance over f defined by[#] :

$$S(I) = \left\{ <uS_1(I_1) \ldots S_n(I_n)> \;\middle|\; \begin{array}{l} \exists <uI_1 \cdots I_n> \text{ in I}, u \models C, \text{ and} \\ \text{for each i in } [1..n],\ S_i(I_i) \models e_i \end{array} \right\}.$$

[#] $Si(I_i) \models e_i$ iff $Si(I_i) \neq \phi$ if $e_i = \exists$, and $Si(I_i) = \phi$ if $e_i = \nexists$.

We now give an example to illustrate the previous definition.

Example 2.3: Let f = COURSE(STUDENT(GRADE)*)*. Consider the two queries :
Q_1: Give the list of math students who got a grade larger than 10, and
Q_2: Give the courses in which some student is registered and did not get any grade for this course.

The query Q_1 is expressed by the expression of selection:

S_1 = COURSE : COURSE=math

(? (STUDENT : ($\exists$(GRADE : GRADE$\geq$10)))).

The query Q_2 is expressed by the expression of selection:

S_2 = COURSE :

($\exists$(STUDENT : ($\nexists$(GRADE)))).

Examples of applications of these two queries are given in Figure 4.

We now present the fourth unary operation, namely restriction. For the sake of simplicity, we shall only consider restrictions on the "root" of the format. It is clear that our definition can be extended to capture more powerful restrictions.

Definition: Let $f \equiv X\,(f_1)^* \ldots (f_n)^*$ be a format for some $n \geq 0$, with $f_1, \ldots, f_n$ non empty, and I an instance over f. A **restriction** on f is an expression of the form $\text{restrict}_{A=B}$ where A and B are in X. The **result** of $\text{restrict}_{A=B}$ **applied to** I, denoted $\text{restrict}_{A=B}(I)$ is defined by :

$$\text{restrict}_{A=B}(I) = \{ \langle uI_1 \cdots I_n \rangle \mid \langle uI_1 \cdots I_n \rangle \text{ in } I, \text{ and } u(A) = u(B) \}.$$

An illustration of the previous definition can be found in Figure 5.

The definition of the last unary operation, namely renaming, is straightforward and thus omitted. An example of renaming can be found in Figure 5.

Clearly, the operations of selection, restriction, and renaming applied to instances over flat format correspond respectively to the relational selection,

EMP	(PHONE	BACK-UP-PH)*
Serge	3537	3468
Nicole	3468	3537
	3329	3329
Francois	3329	3329

Instance J

EMP	(PHONE	BACK-UP-PH)*
Nicole	3329	3329
Francois	3329	3329

$\text{restrict}_{\text{PHONE=BACK-UP-PH}}(J)$

EMP	(P1	P2)*
Serge	3537	3468
Nicole	3468	3537
	3329	3329
Francois	3329	3329

$\text{rename}_{\text{EMP(P1 P2)*}}(J)$

- *Figure 5* - Restriction and Renaming.

restriction, and renaming.

We now introduce five binary operations (union, intersection, difference, join, and cartesian product). For all these operations (except for the cartesian product), we propose two equivalent definitions: the first ones use the inclusion relation on Verso instances, and the second ones are constructive definitions. The equivalence of these alternative definitions is straightforward, and can be found in [Bi].

We start by presenting union, intersection, and difference of Verso instances over identical formats. We shall then extend these three operations to instances over not identical but "compatible" formats.

The operation of union allows to "add" the information contents of two instances. Intersection "extracts" the information common to two instances. The third operation, namely difference, "substracts" the information contained in an instance from the information contained in another one.

Definition: Let f be a format, and I, J two instances over f. Then :

The **union** of I and J, denoted $I \oplus J$, is the smallest instance defined over f containing I and J.

The **intersection** of I and J, denoted $I \cap\!\!\!\!\bigcirc J$, is the greatest instance defined over f contained in I and J.

The **difference** of I and J, denoted $I \ominus J$, is the smallest instance defined over f such that its union with J is equal to $I \oplus J$ (i.e., $(I \ominus J) \oplus J = I \oplus J$).

It is easily seen that $I \ominus J$ is included in I.

Examples of applications of these three operations are given in Figure 6.

We now give constructive definitions for the three operations. First the union.

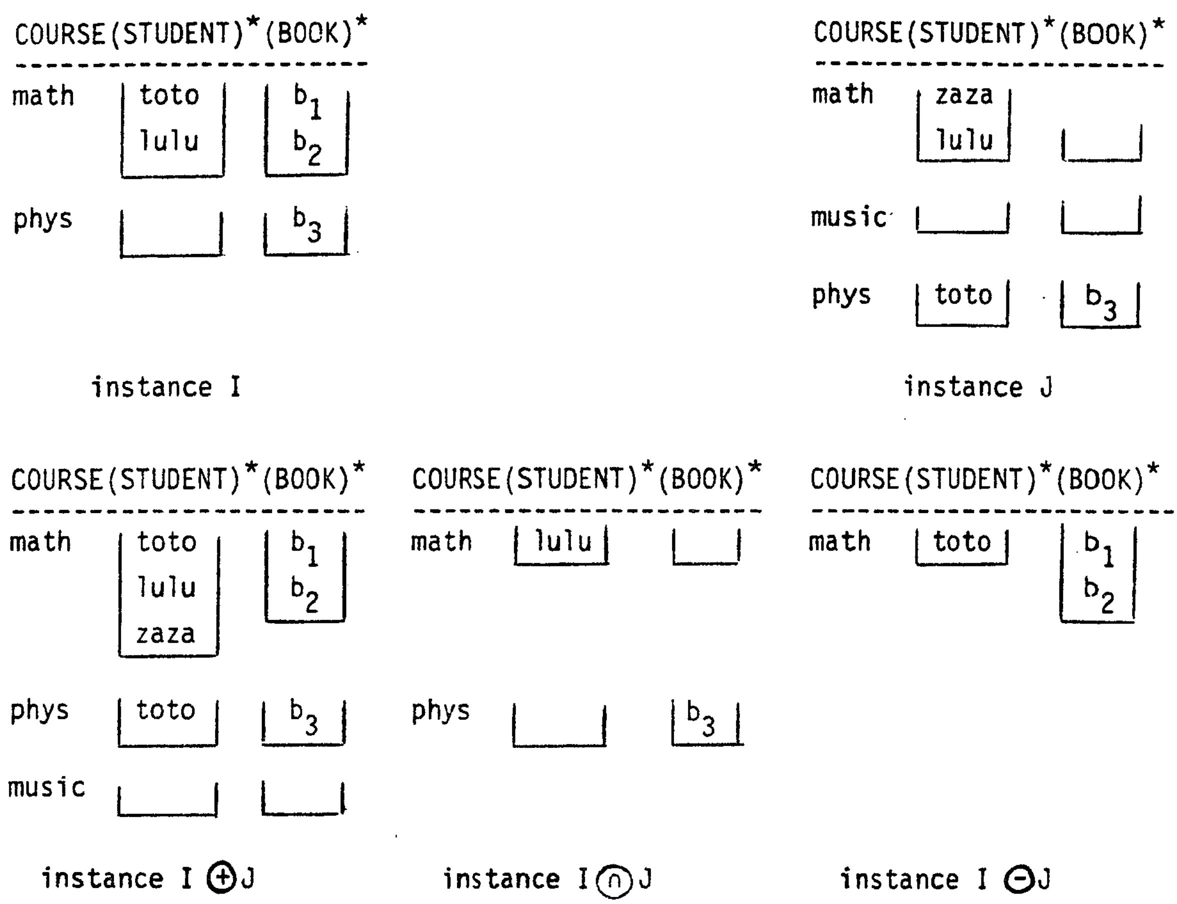
Definition: Let f be a format, and I, J two instances over f. Then the **union** of I and J is the instance over f, denoted $I \oplus J$, recursively defined by :

(i) if $f \equiv X$, X non empty then $I \oplus J = I \cup J$, and

(ii) if $f \equiv X\,(f_1)^* \cdots (f_n)^*$, $f_1, \ldots, f_n$ non empty, then :

$$I \oplus J = \left\{ <u(I_1 \oplus J_1) \cdots (I_n \oplus J_n)> \;\middle|\; \begin{array}{l} <uI_1 \cdots I_n> \text{ in I, and} \\ <uJ_1 \cdots J_n> \text{ in J} \end{array} \right\}$$

$$\cup \left\{ <uI_1 \cdots I_n> \;\middle|\; \begin{array}{l} <uI_1 \cdots I_n> \text{ in I, and} \\ \forall J_1, \ldots, J_n,\ <uJ_1 \cdots J_n> \notin J \end{array} \right\}$$



- *Figure 6* - Binary Operations.

$$\cup \left\{ \langle uJ_1 \cdots J_n \rangle \;\middle|\; \begin{array}{l} \langle uJ_1 \cdots J_n \rangle \text{ in } J, \text{ and} \\ \forall\, I_1, \ldots, I_n,\ \langle uI_1 \cdots I_n \rangle \not\in I \end{array} \right\}.$$

The constructive definition for the intersection is given by:

Definition: Let f be a format, and I, J two instances over f. Then the **intersection** of I and J is the instance over f, denoted I⋒J, recursively defined by :

(i) if $f \equiv X$, X non empty, then $I \circledcap J = I \cap J$, and

(ii) if $f \equiv X\,(f_1)^{*}\ldots(f_n)^{*}$, $f_1, \ldots, f_n$ non empty, then :

$$I \circledcap J = \left\{ \langle u(I_1 \circledcap J_1)\ldots(I_n \circledcap J_n)\rangle \;\middle|\; \begin{array}{l} \langle uI_1 \cdots I_n\rangle \text{ in } I, \text{ and} \\ \langle uJ_1 \cdots J_n\rangle \text{ in } J \end{array} \right\}.$$

The constructive definition for the difference is given by:

Definition: Let f be a format, and I, J two instances over f. Then the **difference** of I and J is the instance over f, denoted $I \ominus J$, recursively defined by :

(i) if $f \equiv X$ then $I \ominus J = I - J$, and

(ii) if $f \equiv X\,(f_1)^{*} \cdots (f_n)^{*}$, $f_1, \ldots, f_n$ non empty, then :

$$I \ominus J = \left\{ \langle u(I_1 \ominus J_1)\ldots(I_n \ominus J_n)\rangle \;\middle|\; \begin{array}{l} \langle uI_1 \cdots I_n\rangle \text{ in } I, \\ \langle uJ_1 \cdots J_n\rangle \text{ in } J, \text{ and} \\ \text{for some } i,\ I_i \ominus J_i \neq \phi \end{array} \right\}$$

$$\cup \left\{ \langle uI_1 \cdots I_n\rangle \;\middle|\; \begin{array}{l} \langle uI_1 \cdots I_n\rangle \text{ in } I, \text{ and} \\ \forall\, J_1, \ldots, J_n\ \langle uJ_1 \cdots J_n\rangle \notin J \end{array} \right\}$$

Note in the example of Figure 6 that the physics COURSE disappeared whereas the math COURSE is still in $I \ominus J$. This result from the condition "$I_i \ominus J_i \neq \phi$" which is true for math and not for physics.

As mentioned earlier, these three operations will be extended to deal with instances over different but compatible formats. To do that, we present the notion of format compatibility.

Definition: Let f and g be two formats respectively over the sets V and W of attributes such that $V \cap W \neq \phi$. Then f and g are **compatible** iff there exists a format h over $V \cup W$ such that f and g are subformats of h.

It can be easily shown that an alternative definition is:

Definition: Let f and g be two formats respectively over the sets V and W of attributes such that $V \cap W \neq \phi$. Then f and g are **compatible** iff there exists a format h' over $V \cap W$ such that h' is a subformat of f and g.

Note that if f and g are compatible, then there is one and only one format h' over $V \cap W$ which is a subformat of both f and g. This unique format is denoted $f \wedge g$.

Now in order to "add" (respectively, "intersect" or "substract") the information contained in an instance I over f, and an instance J over g (f and g compatible), it suffices to extend I and J to a format h such that f and g are both subformats of h, and then to use the union (respectively, intersection, difference).

The union, difference, and intersection according to h are respectively denoted $\oplus_h$, $\ominus_h$, $\textcircled{\cap}_h$. Thus, $I \oplus_h J = I^h \oplus J^h$, $I \ominus_h J = I^h \ominus J^h$, and $I \textcircled{\cap}_h J = I^h \textcircled{\cap} J^h$.

The fourth binary operation, namely join, is directly defined on instances over compatible formats.It allows to "combine" the information contents of two instances.

Definition : Let f and g be two compatible formats respectively defined over the sets of attributes V and W. Let h be a format over $V \cup W$ such that f and g are subformats of h. Let I and J be two instances over f and g respectively. Then the **join** of I and J **according to** h, denoted $I \circledast_h J$, is the greatest instance defined over h, included in $I \oplus_h J$ whose projection on $f \wedge g$ is equal to $I[f \wedge g] \textcircled{\cap} J[f \wedge g]$.

Or, in an equivalent way :

Definition: Let f and g be two compatible formats respectively defined over the sets V and W of attributes. Let h be a format over $V \cup W$ such that f and g are subformats of h. Let I and J be two instances over f and g respectively. Then the **join** of I and J **according to** h is an instance over h, denoted $I \circledast_h J$, recursively defined by :

(i) if $h \equiv X$, X non empty (thus $f \equiv g \equiv h \equiv X$) then $I \circledast_h J = I \cap J$, and

(ii) if $h \equiv X\,(h_1)^*...(h_n)^*$, $h_1, \ldots, h_n$ non empty, $f \equiv X\,(f_1)^*...(f_n)^*$, and $g \equiv X\,(g_1)^*...(g_n)^*$, where for each i, f_i and g_i are subformats of h_i, then :

$$I \circledast_h J = \left\{ <uK_1 \cdots K_n> \;\middle|\; \begin{array}{l} \exists <uI_1 \cdots I_n> \text{ in } I^h, \text{ and } <uJ_1 \cdots J_n> \text{ in } J^h \text{ such that} \\ K_k = I_k \circledast_{h_k} J_k \text{ if } f_k \not\equiv \Lambda,\ g_k \not\equiv \Lambda, \\ K_k = I_k \text{ if } f_k \not\equiv \Lambda,\ g_k \equiv \Lambda, \\ K_k = J_k \text{ if } f_k \equiv \Lambda,\ g_k \not\equiv \Lambda, \end{array} \right\}$$

To illustrate the previous definition, two instances over compatible formats are given in Figure 7, together with their join according to the format $h =$ COURSE(STUDENT)*(BOOK)*.

Note that if f and g are identical formats, the join definition coincides with the intersection definition. The last binary operation, namely cartesian product, is different from the preceding ones in that its first operand is required to be an instance over a flat format.

Definition: Let $f \equiv X$, X non empty, be a flat format and g a format over Y such that $X \cap Y = \phi$. Let I and J be two instances over f and g, respectively. Then the **cartesian product** of I and J, denoted $I \otimes J$, is the instance over $X(g)^*$ defined by :

$$I \otimes J = \{<uJ> \mid u \text{ in } I\}.$$

Note that if f and g are both flat formats, then $I \times J$, and $J \times I$ are different. So the cartesian product is not commutative. Nevertheless, we shall see in Section 4 that the semantics associated with $I \times J$, and $J \times I$ are identical. An example of cartesian product is exhibited in Figure 8.

It should be also noted that the restrictions of union, intersection, difference, over flat formats correspond respectively to the relational union, intersection, difference.

A Verso query is obtained by combining the five binary operations (union, intersection, difference, join and cartesian product), the four unary ones (projection, selection, restriction and renaming) plus an operation which will be

COURSE(STUDENT)*

math	toto zaza
gym	mimi
phys	

"instance I"

COURSE(BOOK)*

math	b_1 b_2
music	b_3
phys	b_4

"instance J"

COURSE(STUDENT)*)(BOOK)*

math	toto zaza	b_1 b_2
phys		b_4

"instance I ⊛$_h$ J"

COURSE(STUDENT)*(BOOK)*

math	toto zaza	b_1 b_2
gym	mimi	
music		b_3
phys		b_4

"instance I ⊕$_h$ J"

COURSE(STUDENT)*(BOOK)*

math	toto zaza	
gym	mimi	

"instance I ⊖$_h$ J"

- *Figure 7* - Compatibility and Binary Operations.

```
UNIVERSITY   DIR.                  COURSE(STUDENT(GRADE)*)*
---------------                    ---------------------------
  ORSAY      Mr X.                 math   | toto    | 10 |  |
                                          |         |  5 |  |
                                          | lulu    |    |  |
                                   phys   | zaza    |  9 |  |

 "instance I"                        "instance J"

UNIVERSITY DIR. (COURSE(STUDENT(GRADE)*)*)*
--------------------------------------------
 ORSAY    Mr X. | math | toto    | 10 |  |
                |      |         |  5 |  |
                |      | lulu            |
                | phys | zaza    |  9 |  |

         "instance I ⊗ J"
```

- *Figure 8* - Cartesian Product.

presented in Section 4, namely restructuring. Together, these operations will be shown to be complete in Section 4.

3. URSA INTERPRETATION OF THE VERSO MODEL

In this section, we exhibit a strong connection between format instances, and relational database instances satisfying the Universal Relation Schema Assumption (URSA). We also give an "interpretation" of the Verso operations in terms of classical relational operations.

In order to do that, we need the notion of format skeleton. Intuitively, the format skeleton of a format f is the relational database schema which describes, in a non hierarchical way, the structure of instances over f.

Definition: Let f be a format. Then the **format skeleton** of f, denoted Skel(f), is the relational database schema recursively defined by:

(i) if $f \equiv X$, X non empty, then Skel(f) = { set(X) }, and

(ii) if $f \equiv X\,(f_1)^* \ldots (f_n)^*$, $f_1, \ldots, f_n$ non empty, then :

Skel(f) = {set(X)} ∪ { set(X)Y | Y in Skel(f_i), for some i in [1..n] }.

For example, the format skeleton of COURSE(STUDENT)*(BOOK)* is the relational database schema { {COURSE}, {COURSE, STUDENT}, {COURSE, BOOK} }. Using these format skeletons, we are now able to "describe" a format instance by a relational database instance.

Definition: Let f be a format, and I an instance over f. The **instance skeleton** of I, denoted skel(I), is the relational database instance over Skel(f) defined by:

(i) if $f \equiv X$, X non empty, then skel(I)(set(X)) = { map(u) | u in I }, and

(ii) if $f \equiv X\,(f_1)^* \ldots (f_n)^*$, $f_1, \ldots, f_n$ non empty, then

$$\text{skel(I)(set(X))} = \{\, \text{map}(u) \mid \langle u I_1 \cdots I_n \rangle \text{ in I for some } I_1, \ldots, I_n \,\}, \text{ and}$$

$$\text{skel(I)(set(X)Y)} = \bigcup_{\langle u I_1 \cdots I_n \rangle \in I} \text{map}(u) * \text{skel}(I_i)(Y)$$

for each i, and each Y in Skel(f_i).

Note in the previous definition that map(u) * skel(I_i) is a relational join operation, and since set(X)∩Y = ϕ, it can also be seen as a cartesian product. However, in the present paper, we use the symbol × to denote ordered cartesian product only. Figure 9 exhibits the instance skeleton of the instance of Figure 1.

We established a correspondance between formats, and relational database schemas (Skel), and between instances over format and relational database instances (skel). It is clear that (1) not all relational database schemas

COUR.
math
phys

COUR.	STUD.
math	toto
math	zaza
phys	lulu
phys	toto

COUR.	STUD.	GRADE
math	toto	4
math	toto	8
phys	lulu	9
phys	toto	6
phys	toto	9

COUR.	BOOK
math	b
math	g

- *Figure 9* - Instance skeleton.

correspond to some formats, and (2) even if a relational database schema R corresponds to a format f, not all instances over R correspond to instances over f.

We shall characterize the "good" (in this context) relational database schemas (Theorem 3.1) and the "good" relational database instances (Theorem 3.2). We first concentrate on relational database schemas.

Theorem 3.1: [Ba2] Let R be relational database schema. Then R is a format skeleton iff :

(1) R is closed under intersection, and

(2) for each X in R, $\{ X \cap Y \mid Y \text{ in } R \}$ is totally ordered by inclusion[#].

We now characterize the relational database instances which are also instance skeletons.

Theorem 3.2: Let f be a format, and R= Skel(f). Let r be an instance over R. Then the following two assertions are equivalent:

(1) r= skel(I) for some I over f, and

[#] A set S is totally ordered by inclusion if for each Z, Z' in S, $Z \subseteq Z'$ or $Z' \subseteq Z$.

(2) r satisfies the URSA.

Proof: For the sake of readability, we use the same notation for an ordered tuple and for the corresponding tuple defined as a mapping.

We first prove that (1) $\Longrightarrow$ (2). The proof is done by induction on the cardinality of Skel(f), denoted #(Skel(f)).

If #(Skel(f))=1 then (1) $\Longrightarrow$ (2) since (2) is always true. Suppose that (1) $\Longrightarrow$ (2) for all f such that #(Skel(f))<m. Let $f \equiv X\,(f_1)^*...(f_n)^*$ be a format with #(Skel(f))=m. Let r be an instance over Skel(f) such that r= skel(I) for some I over f. Let Z_1, Z_2 be in Skel(f) with $Z_1 \subseteq Z_2$. Then Z_2=set(X)Y_2 for some $Y_2 \neq \phi$. Then Y_2 is in Skel(f_j) for some j. Two cases arise:

(a) Z_1 = set(X). Then,

$r(Z_1) = \{ u \mid \langle uI_1 \cdots I_n \rangle \text{ in } I \} \supseteq \{ u \mid \langle uI_1 \cdots I_n \rangle \text{ in } I \text{ and } I_j \neq \phi \} = \pi_{Z_1}(r(Z_2))$

Therefore $r(Z_1) \supseteq \pi_{Z_1}(r(Z_2))$.

(b) Z_1 = set(X)Y_1 for some Y_1 in Skel(f_i) for some i. Since $Z_1 \subseteq Z_2$, it is easily seen that i=j and $Y_1 \subseteq Y_2$. By the induction hypothesis, #(Skel(f_i))<m, and thus $\text{skel}(I_j)(Y_1) \supseteq \pi_{Y_1}(\text{skel}(I_j)(Y_2))$ for each instance I_j over f_j.

$$r(Z_1) = \bigcup_{\langle uI_1 \cdots I_n \rangle \text{ in } I} u * \text{skel}(I_j)(Y_1)$$

$$\supseteq \bigcup_{\langle uI_1 \cdots I_n \rangle \text{ in } I} u * \pi_{Y_1}(\text{skel}(I_j)(Y_2))$$

$$\supseteq \pi_{Y_1}\left(\bigcup_{\langle uI_1 \cdots I_n \rangle \text{ in } I} u * \text{skel}(I_j)(Y_2) \right)$$

$$\supseteq \pi_{Y_1 \text{set}(X)}(r(Z_2)) = \pi_{Z_1}(r(Z_2)).$$

Thus $r(Z_1) \supseteq \pi_{Z_1}(r(Z_2))$ in each case. Hence r satisfies the URSA, so (1) $\Longrightarrow$ (2).

To prove that (2) $\Longrightarrow$ (1), it suffices to exhibit for each r over R satisfying the URSA, an instance I over f such that r=skel(I). Indeed, we now present a recursive algorithm which computes such an instance.

Algorithm 3.1 :
Input : a format f, and an instance r over Skel(f) satisfying the URSA.
Output : I(f,r) a Verso instance defined over f.
begin

   if $f \equiv X$ **then** $I(f,r) = \{u \mid u \text{ in } Otup(X) \text{ and } map(u) \text{ in } r(X)\}$.
   if $f \equiv X\,(f_1)^* \ldots (f_n)^*$ **then**
   begin

      for each x in r(set(X)) and i in [1..n] ,
      let $C = \bigwedge_{A \in set(X)} [A = x(A)]$, and
      let r(i,x) be the relational database instance over Skel(f_i) defined by:
      $r(i,x)(Y) = \pi_Y\,[select_{[C]}(r(set(X)Y))]$ for each Y in Skel(f_i) then
      $$I(f,r) = \left\{ <uI(f_1,r(1,x)) \cdots I(f_n,r(n,x))> \;\middle|\; \begin{array}{l}\text{for some } x \text{ in } r(X), \\ u \text{ in } Otup(X),\ x = map(u)\end{array} \right\}$$
   end

end

One can easily prove by induction that r=skel(I(f,r)). Hence (2) $\Longrightarrow$ (1) which concludes the proof. ▫

By the previous theorem, skel is a mapping from instances over f into relational database instances over Skel(f) satisfying the URSA. Therefore, it would be interesting to characterize Verso operations on instances in terms of relational operations on relational database instances.

Indeed this is the purpose of our next result. In order to prove it, we need some notation and one lemma.

Notation : Let r and s be two relational database instances over the same database schema R. Then $r \subseteq s$ iff $r(X) \subseteq s(X)$ for each X in R. Also $r \cup s$ is the relational database instance over R defined by $(r \cup s)(X) = r(X) \cup s(X)$ for each X in R. Finally, $r \cap s$ and $r - s$ are defined in a similar way.

The lemma that we shall use relates containment of Verso instances to containment of the corresponding instance skeletons. Formally, we have:

Lemma 3.1 : Let f be a format, and I, J two instances over f. Then $I \leq J$ iff $skel(I) \subseteq skel(J)$.

Proof : First suppose that $I \leq J$. Then by inspection of the definition of an instance skeleton, it is clear that $skel(I) \subseteq skel(J)$.

Now suppose that $skel(I) \subseteq skel(J)$. Then by inspection of Algorithm 3.1 we have: $I = I(f,skel(I)) \leq I(f,skel(J)) = J$. Thus $I \leq J$. ▫

We are now ready to characterize Verso-operations on format instances in terms of relational operations on the corresponding relational database instances.

Theorem 3.3 : Let f,g be two compatible formats, and h a format such that f and g are subformats of h. Let I and J be instances over f and g respectively. Let $r = skel(I^h)$ and $s = skel(J^h)$. Then :

(1) $skel(I \oplus_h J) = r \cup s$,

(2) $skel(I \circledcap_h J) = r \cap s$,

(3) $skel(I \ominus_h J)$ is the smallest URSA-instance over Skel(h) containing $r - s$, and

(4) $skel(I \circledast_h J)$ is the greatest URSA-instance over Skel(h) contained in the instance t over Skel(h) defined by:

(a) $t(X) = r(X) \cap s(X)$ if $X \in Skel(f) \cap Skel(g)$.

(b) $t(X) = r(X)$ if $X \in Skel(f) - Skel(g)$,

(c) $t(X) = s(X)$ if $X \in Skel(g) - Skel(f)$, and

(d) $t(X) = \phi$ otherwise.

Proof : (1) By definition, $I \oplus_h J$ contains I^h and J^h. By Lemma 3.1, $skel(I \oplus_h J) \supseteq skel(I^h) = r$ and $skel(I \oplus_h J) \supseteq skel(J^h) = s$. Hence (+)

$skel(I \oplus_h J) \supseteq r \cup s$.

Since r and s are URSA insances, it is clear that $r \cup s$ is also an URSA instance. By Theorem 3.2, $r \cup s = skel(K)$ for some format instance K over h. By Lemma 3.1, $I^h \leq K$ and $J^h \leq K$. By definition of union , $I^h \oplus J^h \leq K$. Hence (++) $skel(I \oplus_h J) \subseteq skel(K) = r \cup s$. By (+) and (++) $r \cup s = skel(I \oplus_h J)$.

(2) is proved in a similar way.

(3) Let T = { t | t is an URSA instance over Skel(h) containing r-s }. Consider then $t_0 = \bigcap_{t \in T} t$. By Theorem 3.2, $skel(I \ominus_h J)$ is an URSA instance over Skel(h). Since $(I \ominus_h J) \oplus_h J = I \oplus_h J$, $skel(I \ominus_h J) \cup s = r \cup s$. Thus $skel(I \ominus_h J) \supseteq r-s$. Hence $t_0 \subseteq skel(I \ominus_h J)$. Clearly, t_0 is an URSA instance. By Theorem 3.2, $t_0 = skel(K)$ for some format instance K over h. Also, $r-s \subseteq t_0$. Thus $r \cup s \subseteq t_0 \cup s$. Therefore $I^h \oplus J^h \leq K \oplus J^h$ by Lemma 3.1. Since $t_0 \subseteq skel(I \ominus_h J)$, $K \leq I \ominus_h J$ by Lemma 3.1. Hence $K \oplus J^h \leq (I \ominus_h J) \oplus J^h = I^h \oplus J^h$. Thus $K \oplus J^h = I^h \oplus J^h$. By definition of the Verso difference, $I \ominus_h J \leq K$. Since $K \leq I \ominus_h J$ and $I \ominus_h J \leq K$, $K = I \ominus_h J$. Hence $skel(I \ominus_h J) = skel(K) = t_0$, that is the smallest URSA instance over skel(h) containing $r - s$.

(4) Let t_1 be the greatest URSA instance over skel(h) contained in t. By Theorem 3.2, $t_1 = skel(K)$ for some format instance K over h. Since $t_1 \subseteq t \subseteq r \cup s$, $K \leq I \oplus_h J$ by (1). Clearly, $K[f \wedge g] = I[f \wedge g] \cap J[f \wedge g]$. Thus (†) $K \leq I \circledast_h J$ by definition of join.

By Theorem 3.2, $skel(I \circledast_h J)$ is an URSA instance. Clearly, $skel(I \circledast_h J) \subseteq t$. Thus $skel(I \circledast_h J) \subseteq t_1$. Therefore (††) $I \circledast_h J \leq K$ by Lemma 3.1.

By (†) and (††), $I \circledast_h J = K$ which concludes the proof of the theorem. ▫

As shown in Theorem 3.3, it is possible to characterize the binary Verso operations on format instances in terms of relational operations on the corresponding database instances. Furthermore, a constructive characterization can be obtained. This constructive characterization can be found in [Bi] and allows to compute $skel(I \oplus_h J)$, $skel(I \bigcirc_h J)$, $skel(I \ominus_h J)$ and $skel(I \circledast_h J)$ from skel(I) and skel(J) where I and J are instances over f and g respectively, f and g compatible and subformats of h. Finally, it is also possible to characterize unary Verso operations on format instances in terms of relational operations on the corresponding relational database instances (see [Bi]).

4. Data restructuring

In this section, we introduce the last unary Verso operation, namely restructuring. This operation allows one to modify the data structure used to store information. When transforming an instance over some format g into an instance over another format f, we may loose some information. In order to study this, we first formalize the notion of information contained in an instance. We then define the data restructuring operation based on a principle of minimum loss of information. We then characterize the properties which must be satisfied by f and g to allow data restructuring of *all* instances over g into instances over f without loss of information. Finally, we study the dependencies that *some* instances over some format g satisfy, so that data restructuring according to some format f is possible without loss of information.

We first try to capture the semantics of Verso instances using the notion of "facts". In this context, a fact is a tuple, and it is also the elementary unit of information.

Two basic operations on sets of facts are considered. They are: the closure under projection and under join.

Definition : Let H be a set of facts. Then the **closure of** H **under projection,** denoted $\Pi(H)$, is defined by :

$$\Pi(H) = \{ \pi_Y(x) \mid x \text{ in } H \cap Tup(X) \text{ for some } X \text{ and } Y \subseteq X \},$$

and the **closure of** H **under join**, denoted $*H$, is defined by:
For each $n \geq 0$, let H_n be obtained by:

- $H_0 = H$,

- $H_{i+1} = \{ x * y \mid x \in H, y \in H_i, x \text{ and } y \text{ joinable} \}$.

Then $*H = \bigcup_{i=0..\infty} H_i$.

Now, given a set of facts, it seems reasonable to deduce new facts by projection of known facts. The closure under join is already more arguable. For instance, if "toto" is taking "math" and "math" is taught by "Miss Jones", you do not want to conclude that "Miss Jones" is teaching "math" to "toto". The semantics that we are going to associate with format instances states that the "legal" joins are only the joins of tuples in the instance skeletons. More formally, we have:

Definition : Let I be an instance over the format f. Then the **set of facts associated with** I, denoted fact(I), is defined by:

$$\text{fact}(I) = \Pi(*(\bigcup_{Z \text{ in } Skel(f)} \text{skel}(I)(Z))) .$$

The previous definition is illustrated in Figure 10 where the set of facts associated with the instance I of Figure 1 is given.

The notion of set of facts associated with a format instance is used now to present the last unary operation, namely restructuring.

```
<math, toto, 4, b>, <math, toto, 8, b>
<math, toto, 4, g>, <math, toto, 8, g>
<math, toto, 4>,..., <math, 4>,..., <4>,
<math, toto, g>,..., <math, g>,..., <g>,
<phys, lulu, 9>,..., <phys, lulu>,..., <9>,...
```

- *Figure 10* - Fact(I).

Definition : Let f be a format. Let J be an instance over some format. Then **the result of restructuring** J **according to** f, denoted $restruct_{[f]}(J)$, is the greatest[#] instance I defined over f such that fact(I) $\subseteq$ fact(J).

To illustrate this definition, we present in Figure 11 an instance J over the format COURSE(STUDENT GRADE)* and the results I_1 and I_2 of restructuring J according to $f_1 =$ COURSE(STUDENT(GRADE)*)* and $f_2 =$ STUDENT GRADE (COURSE)*. Note that the instance I_1 contains the same information than the instance J, but since no STUDENT is registered in the music COURSE in J, the fact that there exists a music COURSE, has been lost in I_2.

Now the following problem arises : Let J be an instance over some format g, and let f be a format. Is $restruct_{[f]}$ a non-loss operation for J? In other words, is fact(J) = fact($restruct_{[f]}(J)$) ?

We first adress the case when it is always possible to represent an instance over g by an instance over f, i.e. $restruct_{[f]}$ is non-loss for all instances over g.

In order to do that, we need a way to compare the representative power of formats. Formally:

Notation : Let f be a format. Then SAT(f)= { fact(I) | I in Inst(f) }.

Definition : Let f and g be two formats. Then f is **dominated by** g, denoted $f \leq g$, iff SAT(f) $\subseteq$ SAT(g). Also f and g are **equivalent**, denoted $f \equiv g$, iff $f \leq g$ and $g \leq f$ (i.e. SAT(f)=SAT(g)).

Intuitively, f is dominated by g iff each instance over f can be represented by an instance over g containing the same information. Two characterizations of format dominance are now presented. The first one (Lemma 4.1) is based on properties of the corresponding format skeletons. The second one (Theorem 4.1) is based on some elementary format transformations. We now present the first characterization of format dominance.

[#] It is clear that there exists a finite number of instances I such that fact(I) $\subseteq$ fact(J). Then $restruct_{[f]}(J)$ is obtained by union of these instances.

COURSE(STUDENT GRADE)*

COURSE	STUDENT	GRADE
math	toto	10
	toto	5
	lulu	3
music		
phys	toto	10

Instance J

COURSE(STUDENT(GRADE)*)*

COURSE	STUDENT	(GRADE)*
math	toto	10, 5
	lulu	3
music		
phys	toto	10

$\text{restruct}_{[f_1]}(J) = I_1$

STUDENT GRADE(COURSE)*

STUDENT	GRADE	(COURSE)*
toto	10	math, phys
toto	5	math
lulu	3	math

$\text{restruct}_{[f_2]}(J) = I_2$

- *Figure 11* - Restructuring.

Lemma 4.1 : Let f and g be two formats. Then $f \leq g$ iff $Skel(f) \subseteq Skel(g)$. Thus $f \equiv g$ iff $Skel(f) = Skel(g)$.

Proof : First suppose that $f \leq g$. Let X be in Skel(f). For each A in X, let c_A and d_A be two distinct values in dom(A). Let s be the relational database instance over Skel(f) defined by:

(1) $s(X) = \left\{ x \;\middle|\; \begin{array}{l} \text{for each A in X, } x(A) = c_A \text{ or } x(A) = d_A \;, \\ \text{for some A in X, } x(A) = d_A \end{array} \right\}$

(2) $s(Y) = \pi_Y(s(X))$ if $Y \subseteq X$, and

(3) $s(Y) = \phi$ otherwise.

It is clear that s is an URSA instance over Skel(f). By Theorem 3.2, s is an instance skeleton. Hence $\Pi(*(\bigcup_{Z\in Skel(f)} s(Z)))$ is in SAT(f). Hence $\Pi(*(\bigcup_{Z\in Skel(f)} s(Z))) = \Pi(s(X))$ is in SAT(f). Since $f \le g$, $SAT(f) \subseteq SAT(g)$, so there exists an instance skeleton r over Skel(g) such that $\Pi(s(X)) = \Pi(*(\bigcup_{Z\in Skel(g)} r(Z)))$. Let x be in s(X). Then x is in $\Pi(*(\bigcup_{Z\in Skel(g)} r(Z)))$. Thus there exists a sequence $Z_1, \ldots, Z_n$ of attribute sets in Skel(g), and a sequence $z_1, \ldots, z_n$ of facts such that $z_j \in Tup(Z_j)$ for each j and $x = \pi_X(\underset{j=1..n}{*} z_j)$. Suppose that X is not one of $Z_1, \ldots, Z_n$. Clearly, for each j, $z_j \in \Pi(s(X))$, so $Z_j \subseteq X$. Let x_0 be the tuple over X defined by $x_0(A) = c_A$ for each A in X. Note that $x_0 \notin \Pi(s(X))$. For each j, let z_j' be a tuple in s(X) such that $\pi_{Z_j}(x_0) = \pi_{Z_j}(z_j')$. (Such a tuple clearly exists by construction of s). Hence $x_0 = \pi_X(\underset{j=1..n}{*} z_j')$ is in $\Pi(*(\bigcup_{Z\in Skel(g)} r(Z))) = \Pi(s(X))$. Thus x_0 is in s(X), a contradiction with the definition of s. Hence X is one of $Z_1, \ldots, Z_n$. Therefore X is in Skel(g). Thus $Skel(f) \subseteq Skel(g)$.

Now suppose that $Skel(f) \subseteq Skel(g)$. Let H be in SAT(f). Then $H = \Pi(*(\bigcup_{Z\in Skel(f)} skel(I)(Z)))$ for some instance I over f. Consider the relational database instance s over Skel(g) defined by :

(a) $s(X) = skel(I)(X)$ if $X \in Skel(f) \cap Skel(g)$,

(b) $s(X) = \bigcup_{\substack{Y \supseteq X \\ Y\in Skel(f)}} \pi_X(s(Y))$ if $X \in Skel(g) - Skel(f)$, and

(c) $s(X) = \phi$ otherwise

By Theorem 3.2, s is an instance skeleton over Skel(g). Thus s = skel(J) for some instance J over g. It is easily seen that $H = \Pi(*(\bigcup_{Z\in Skel(g)} skel(J)(Z)))$. Hence H is in SAT(g). Therefore $SAT(f) \subseteq SAT(g)$ and so $f \le g$. ▫

In order to present the second characterization of format dominance, we exhibit three format transformations. These transformations are presented in their elementary versions and then generalized.

Definition :

(a) Let $f \equiv X\,(f_1)^* \ldots (f_n)^*$, and $g \equiv Y\,(g_1)^* \ldots (g_n)^*$, $f_1,\ldots,f_n$, $g_1,\ldots,g_n$ non empty, then g is obtained from f by **elementary root permutation** iff :

(i) $f_i = g_i$ for each i in $[1..n]$, and

(ii) set(X) = set(Y).

g is obtained from f by **elementary branch permutation** iff :

(i) for each i in $[1..n]$, there exists j in $[1..n]$ such that $g_j = f_i$, and

(ii) X=Y.

(b) Let $f \equiv XY\,(f_1)^* \ldots (f_n)^*$ and $g \equiv X\,(Y\,(f_1)^* \ldots (f_n)^*)^*$ then g is obtained from f by **elementary compaction**.

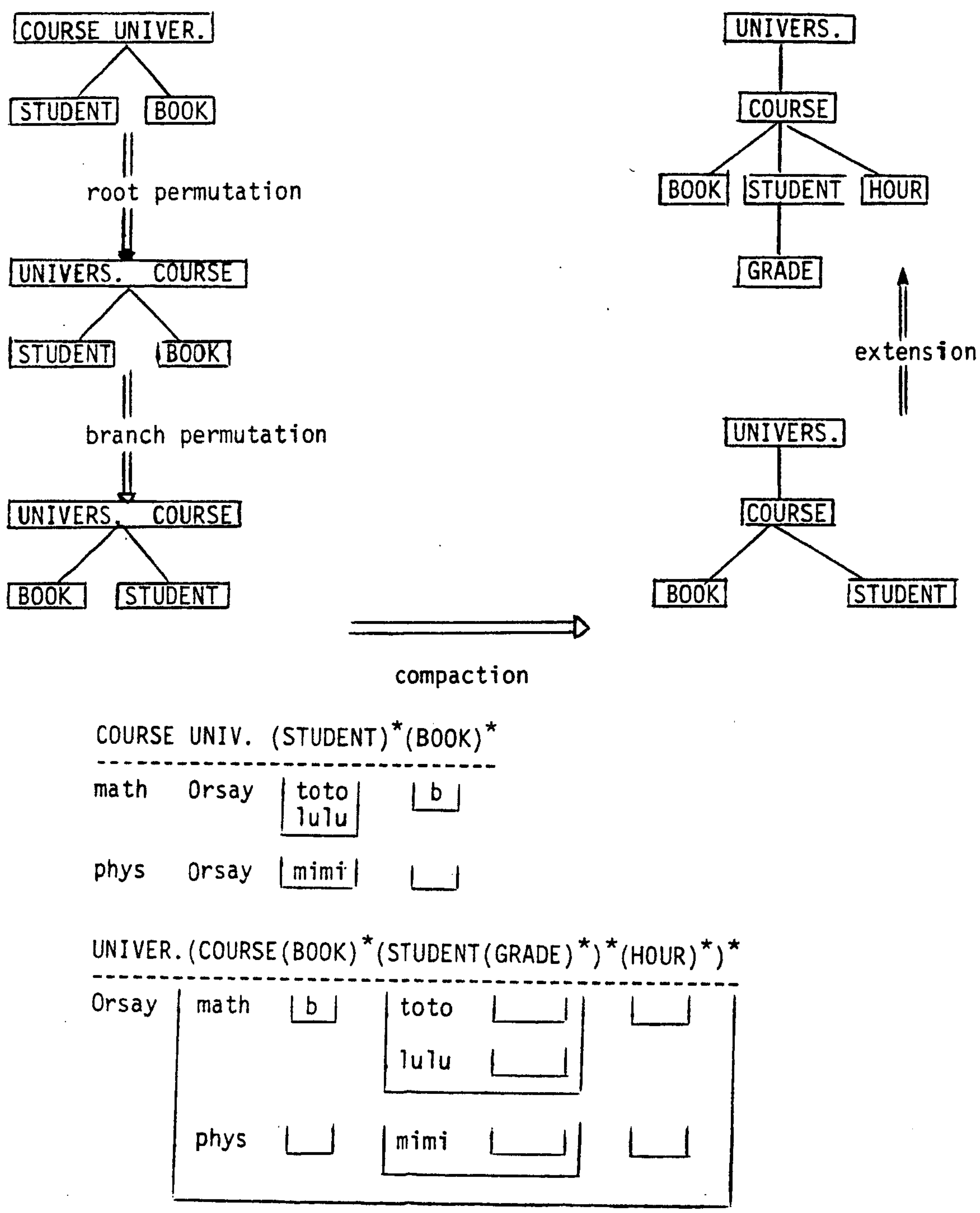
Now a root permutation on a format f is obtained by applying elementary root permutations to components f_i of $f \equiv X\,(f_1)^* \ldots (f_n)^*$. Branch permutation and compaction are obtained from elementary branch permutation and elementary compaction in a similar manner. Figure 12 exhibits a sequence of these three transformations together with the extension defined in Section 3.

We are now ready for a second characterization of format dominance and equivalence (The proof follows easily from Lemma 4.1).

Theorem 4.2 : Let f and g be two formats. Then $f \equiv g$ iff g can be obtained from f by a finite sequence of root and branch permutations. Also $f \leq g$ iff g can be obtained from f by a finite sequence of root and branch permutations, compactions and extensions.

Even if f is not dominated by g, some particular instances over f are representable by instances over g without loss of information. That is because those particular instances satisfy some constraints on top of the constraints that are implied by the format f. We now define two kinds of dependencies which are going to capture these constraints.

In order to do that, we need the following notation.



- *Figure 12* - Format transformations (Part I and II)

Notation : Let H be a set of facts, X a relational schema and R a relational database schema. Then :

- $H_{|X} = \{ x \mid x \in H \cap tup(X) \}$, and
- $H_{|R} = \bigcup_{X \in R} H_{|X}$.

Now we have:

Definition : Let R be a relational database schema, $Z = \bigcup_{Z \in R} X$ and H a set of facts. Then *R denotes the **schema join dependency** (SJD) **associated with** R, and H **satisfies** *R, denoted $H \models *R$, iff $H_{|Z} = (* [H_{|R}])_{|Z}$.

Also $\exists$ R denotes the **schema existence dependency** (SED) **associated with** R, and H **satisfies** $\exists$, denoted $H \models \exists R$, iff $H = \Pi(H_{|R})$.

An example is now given to motivate the use of the word "existence" for name of the second kind of dependency in the above definition.

Example 4.1 : Consider the formats f = COURSE(STUDENT $)^*$ and g = STUDENT(COURSE $)^*$, and the instances I, J of Figure 13. Then J = $restruct_{[g]}(I)$, and fact(J) = fact(I)$-${<phys>}. When restructuring I, we lost the "phys." COURSE because there is no STUDENT registered in this COURSE in I. In other words, we lost some information because fact(I) $\not\models \exists\{$ {COURSE,STUDENT} $\}$.

The next result uses the previous dependencies to characterize the sets of facts which are representable by instances over a given format.

Theorem 4.3 : Let f be a format and H a set of facts. Then H can be represented by an instance over f (i.e. there exists an instance I over f such that H = fact(I)) iff:

COURSE(STUDENT)*	
math	toto lulu zaza
music	toto zaza
phys	

Instance I

STUDENT(COURSE)*	
toto	math music
lulu	math
zaza	math music

$J = restrict_g(I)$

- *Figure 13* - Existence Dependency.

(i) $H \models {}^*R$ for each $R \subseteq Skel(f)$, and

(ii) $H \models \exists S$ where $S = \{ \bigcup_{Y \in R} Y \mid$ for some $R \subseteq Skel(f) \}$.

Proof : First suppose that H can be represented by some instance I over f. Then $fact(I) = H$. Thus : $H = \Pi({}^*(\bigcup_{X \in Skel(f)} skel(I)(X)))$. Let $Z = \bigcup_{X \in R} X$. Let u be in $H_{|Z}$. Clearly, $u = \mathop{*}_{X \in R} u_X$ where u_X is in skel(I)(X) for each X in R. Hence u is in ${}^*[H_{|R}]$. Therefore u is in $({}^*[H_{|R}])_{|Z}$. Thus $H_{|Z} \subseteq ({}^*[H_{|R}])_{|Z}$. A similar argument shows that the converse inclusion is also true. Hence $H \models {}^*R$, and so (i) is verified.

By definition of S,

$$\Pi(H_{|S}) = (\Pi (\Pi({}^*(\bigcup_{X \in Skel(f)} skel(I)(X)))))_{|S}$$

$$= \Pi (({}^*(\bigcup_{X \in Skel(f)} skel(I)(X)))_{|S})$$

$$= \Pi(*(\bigcup_{X \in Skel(f)} skel(I)(X)))$$

$$= H.$$

Therefore, $H \models \exists S$, so (ii) is verified.

Now suppose that $H \models *R$ and $H \models \exists S$. Let r be the relational database instance over Skel(f) defined by $r(X) = H_{|X}$ for each X in Skel(f). It is easily seen that r is an URSA instance and by Theorem 3.2, there exists I instance over f such that r = skel(I) and fact(I) = H. Hence H can be represented by an instance over f which concludes the proof. ▫

Now we have:

Corrolary : Let f be a format. Then $restruct_{[f]}$ is without lost of information for an instance I iff :

(i) $skel(I) \models *R$ for each $R \subseteq Skel(f)$, and

(ii) $skel(I) \models \exists S$ where $S = \{ \bigcup_{Y \in R} Y \mid \text{for some } R \subseteq Skel(f) \}$. ▫

Restructuring is the last operation of the Verso algebra. Together with the five binary operations, and four unary ones already presented they are as powerfull as the relational algebra. More precisely, we have :

Theorem 4.4 : Let Σ be a Verso schema, such that $Skel(f) \cap Skel(g) = \phi$ for each f and g in Σ. Let $R = \bigcup_{f \in \Sigma} Skel(f)$ be the corresponding relational database schema. Then for each relational query α over R, there exists a Verso query β (with flat target format) over Σ, such that $\alpha(r) = map(\beta(I))$ for each instance I over Σ and relational database instance $r = skel(I)$ over R.

Proof : (Sketch) The base relations of r can be obtained from I using a Verso selection followed by a projection. The relational projection, restriction, selection, renaming, union, difference, intersection and cartesian product are simulated respectively by the Verso projection, restriction, selection, renaming, union, difference, intersection and cartesian product. The restructuring may be necessary in order to apply these operations. ▫

Remark 4.1 : The relational join can be realized using other relational operations (renaming, cartesian product, restriction and projection), and thus can be simulated using the corresponding Verso operations. However, a simpler and more natural way to simulate the relational join is to use a restructuring followed by a Verso join. This remark is illustrated in Figure 14. In order to do a relational join of r_1 over {COURSE STUDENT} and r_2 over {COURSE BOOK}, r_1 is restructured according to COURSE(STUDENT)*, r_2 according to COURSE(BOOK)*. Then a Verso join is performed.

COURSE	STUDENT
math	toto
math	zaza
phys	lulu
phys	mimi

COURSE	BOOK
math	b_1
math	b_2
music	b_3
phys	b_4

COURSE(STUDENT)*(BOOK)*		
math	toto zaza	b_1 b_2
phys	lulu mimi	b_4

- *Figure 14* - Simulating the relational join.

5. Expressive power of Verso selection

In the previous section, we showed that the Verso operations are "complete" (i.e. they are at least as powerfull as the relational operations). In this section, we discuss the expressive power of the selection. We then introduce an extension of the selection, and exhibit a very large set of relational queries which can be simulated by a "super"-selection followed by a projection.

We first present a query which would typically require a join in the relational model but can be simply expressed by a selection in the Verso model.

Example 5.1 : Consider the format f = $COURSE(STUDENT)^*(EXAM\text{-}DAY)^*$. Now consider the query : "What are the COURSEs taken by the STUDENT toto which have an EXAM-DAY on November first ? ". In the relational model, there would typically be two relational schemas {COURSE STUDENT} and {COURSE EXAM-DAY} and the query would require a join operation. This query can be answered by the Verso selection :

$$S \equiv COURSE : (\exists(STUDENT : STUDENT = toto),$$

$$\exists(EXAM\text{-}DAY : EXAM\text{-}DAY = November 1st)).$$

Indeed, some very natural queries like "Give the list of COURSEs with no known EXAM-DAY ? " can be answered by a Verso selection whereas they would require the use of difference in the pure relational model.

We now propose a simple extension of the Verso selection which dramatically increases its power. Let us consider the following query on $COURSE(STUDENT(GRADE)^*)^*$: "Give the list of COURSEs, STUDENTs and GRADEs such that toto got an A in the COURSE and a STUDENT (not necessarily toto) got an F in the COURSE". It should be noted that this query is complicated by the fact that they are several roles for the same attribute, namely STUDENT. Typically, such a query would require several joins in the classical relational model.

What we mean by such a query is in fact two selections on GRADE, say S_1 = GRADE : GRADE = A and S_2 = GRADE : GRADE = F.

Now we need two selections on STUDENT(GRADE)* :

$S_1' =$ STUDENT : STUDENT = toto ($\exists(S_1)$),and

$S_2' =$ STUDENT : ($\exists$(S2)).

The first one filters toto if he got an A, and the second one any STUDENT who got an F. Now we can express our query by :
S = COURSE : (?(S') | { $\exists(S_1')$, $\exists$(S2') }) where S' is the identity on STUDENT(GRADE)*.

It should be noted that this is not a selection as defined in Section 2. Intuitively, when we perform such a selection on an instance I over COURSE(STUDENT(GRADE)*)*, for each element $<uI_1>$ of I, we perform S_1' and S_2' on I_1 "in parallel" and we write I_1 (i.e. $S'(I_1)$) iff $S_1'(I_1) \neq \phi$ and $S_2'(I_1) \neq \phi$. Note that, in this case, S_1' and S_2' are used exclusively as conditions.

We now formally define the "super"-selection.

Definition : Let $f \equiv X\ (f_1)^* \ldots (f_n)^*$, $f_1, \ldots, f_n$ non empty, be a format for some $n \geq 0$, and I an instance over f. Then a **super-selection** S over f is an expression recursively defined by :

(a) if S is a selection over f, then S is a **super-selection** over f, and

(b) For i=1...n, let S_i be a super-selections over f_i and $\bar{S}_i$ be a finite set of expressions of the form e'(S') where $e' \in \{ \exists, \nexists \}$ and S' is a **super-selection** over f_i.

Then the expression $S \equiv X : C\ (\ e_1(S_1) \mid \bar{S}_1, \ldots, e_n(S_n) \mid \bar{S}_n\ )$ where C is a condition on X, and for i=1...n, $e_i \in \{ \exists, \nexists, ? \}$ is a **super-selection** over f.

The corresponding operation is defined by :

Definition : Let $f \equiv X\ (f_1)^* \ldots (f_n)^*$, $f_1, \ldots, f_n$ non empty, be a format for some $n \geq 0$ and $S \equiv X : C\ (\ e_1(S_1) \mid \bar{S}_1, \ldots, e_n(S_n) \mid \bar{S}_n\ )$ a super-selection over f. Then the **result of** S **applied to** I, denoted S(I), is the instance over f defined by :

$$S(I) = \left\{ \langle uS_1(I_1)\ldots S_n(I_n)\rangle \;\middle|\; \begin{array}{l} \langle uI_1 \cdots I_n\rangle \text{ in } I,\ u \models C, \\ \text{for all } i \text{ in } [1..n],\ S_i(I_i) \models e_i, \text{ and} \\ S'(I_i) \models e' \text{ for each } e'(S') \text{ in } \overline{S}_i \end{array} \right\}$$

It turns out that the super-selection can easily be expressed using the Verso selection, projection and join. To prove this, we need the following lemma.

Lemma 5.1 : Let $f \equiv X\,(f_1)^* \ldots (f_n)^*$ be a format and $S \equiv X : C\,(\,e_1(S_1) \mid \overline{S}_1, \ldots, e_i(S_i) \mid \overline{S}_i \cup \{\,e_0(S_0)\,\}, \ldots, e_n(S_n) \mid \overline{S}_n)$ a super-selection over f. Then for each I in Inst(f) : $S(I) = S'(I) \circledast_f (S''(I)[X])$ where :

$S' \equiv X : C\,(\,e_1(S_1) \mid \overline{S}_1, \ldots, e_i(S_i) \mid \overline{S}_i, \ldots, e_n(S_n) \mid \overline{S}_n\,)$, and

$S'' \equiv X : C\,(\;?(Id_{I_1}), \ldots, ?(Id_{I_{i-1}}), e_0(S_0), ?(Id_{I_{i+1}}), \ldots, ?(Id_{I_n})\,)$.

Proof : Let I be an instance over f. Let $J = S(I)$ and $K = S'(I) \circledast_f (S''(I)[X])$. Then ω is in J iff $\omega = \langle uS_1(I_1) \cdots S_n(I_n)\rangle$ for some $\langle uI_1 \cdots I_n\rangle$ in I satisfying :

(i) $u \models C$,

(ii) for each j in [1..n], $S_j(I_j) \models e_j$,

(iii) for each j in [1..n] and $e'(S')$ in $\overline{S}_j$, $S'(I_j) \models e'$, and

(iv) $S_0(I_i) \models e_0$.

Hence ω is in J iff ω is in S'(I) and $\omega[X]$ is in S''(I). Therefore ω is in J iff ω is in K which concludes the proof. ▫

Using Lemma 5.1, one can easily show :

Theorem 5.1 : For each super-selection β, there exists a Verso query β' composed exclusively of selections, projections and joins such that $\beta = \beta'$.

We now present a large class of relational queries which can be simulated using a super-selection followed by a projection. Intuitively, these queries are all the queries obtained using relational selections, joins and projections such that the projections do not violate the underlying structure of the corresponding Verso instance.

Theorem 5.2 : Let f be a format, R = Skel(f) the corresponding relational database schema. Let q be a (relational) selection-projection-join query on R such that every projection in q is a projection on some union of attribute sets in R. Then there exists a Verso query q' consisting of a (Verso) super-selection followed by a (Verso) projection, such that q' is equivalent to q.

Proof : (Sketch) The proof is done by induction on the depth of q. As mentioned in the proof of Theorem 4.4, the base relations can be obtained using a Verso (simple) selection followed by a Verso projection. Thus the Theorem is true for queries of depth 1.

Now let q_1 and q_2 be two relational queries respectively equivalent to $S_1[f_1]$ and $S_2[f_2]$ where $[f_1],[f_2]$ are Verso projections, S_1,S_2 are Verso super-selections. Three cases have to be considered :

(a) Let $q = \pi_X(q_1)$ where X is the union of attribute sets in R. Clearly, X must be also included in $\bigcup_{Y \in Skel(f_1)} Y$. Then there exists a subformat g of f_1 such that $X = \bigcup_{Y \in Skel(g)} Y$. Thus $\pi_X(q_1) = q$ is equivalent to $(S_1[f_1])[g]$. Since $(S_1[f_1])[g] = S_1[g]$, $q = \pi_X(q_1)$ is equivalent to a super-selection followed by a projection.

(b) Let $q = select_{[C]}(q_1)$. Then some extra conditions can clearly be introduced in S_1 to obtain S such that $q = select_{[C]}(q_1)$ is equivalent to $S[f_1]$.

(c) Let $q = q_1 * q_2$. Clearly, f_1 and f_2 are subformats of f. Thus there exists a subformat g of f such that $Skel(g) = Skel(f_1) \cup Skel(f_2)$. Since we allow in a super-selection several selections to occur on the same sub-instance concurrently, we can combine the selections used to build S_1 and S_2 to obtain a super-selection S such that $q = q_1 * q_2$ is equivalent to $S[g]$. □

We now illustrate the previous theorem.

Example 5.2 : Consider the query : "List all COURSEs attended by both the STUDENTs toto and lulu, and for each of these COURSEs, list the STUDENTs in that COURSE". This query corresponds to the following relational query over the database schema R = {COURSE, STUDENT} :

$$\pi_{COURSE,\ STUDENT}(R) * \pi_{COURSE}select_{[STUDENT = toto]}(R) * \pi_{COURSE}select_{[STUDENT = lulu]}(R).$$

This relational query can be decomposed as follows :

q_1 = [COURSE STUDENT],
$q_2 = select_{[STUDENT = toto]}(q_1)$,
$q_3 = \pi_{COURSE}(q_2)$,
$q_4 = select_{[STUDENT = lulu]}(q_1)$,
$q_5 = \pi_{COURSE}(q_4)$,
$q_6 = q_3 * q_5$, and
$q_7 = q_1 * q_6$.

We now follow the construction sketched in the proof of the theorem to obtain an equivalent Verso query formed of a super-selection followed by a projection. Let g be the format COURSE(STUDENT)*. For each i in [1..7], Q_i defined below is equivalent to q_i.

Q_1 = (COURSE($\exists$ (STUDENT))[g],
Q_2 = (COURSE($\exists$ (STUDENT :STUDENT = toto))[g],
Q_3 = (COURSE($\exists$ (STUDENT :STUDENT = toto))[COURSE]
Q_4 = (COURSE($\exists$ (STUDENT : STUDENT = lulu))[g],
Q_5 = (COURSE($\exists$ (STUDENT : STUDENT = lulu))[COURSE],

$$Q_6 = (\text{COURSE}(\ ?(\text{STUDENT} : \left| \begin{Bmatrix} \exists(\text{STUDENT :STUDENT = toto}), \\ \exists(\text{STUDENT :STUDENT = lulu}) \end{Bmatrix} \right)\ ) \ [\text{COURSE}], \text{ and}$$

$$Q_7 = (\text{COURSE}(\ ?(\text{STUDENT} : \left| \begin{Bmatrix} \exists(\text{STUDENT :STUDENT = toto}), \\ \exists(\text{STUDENT :STUDENT = lulu}) \end{Bmatrix} \right)\ )\ .$$

REFERENCES

[AMM]

H. Arisawa, K. Moriya, T. Miura, "Operations and Properties on Non-First-Normal-Form Relational Databases", Proc. Inter. Conf. on VLDB, Florence, 1983, pp 197-204.

[Ba1]

F. Bancilhon & all, "Verso : A Relational Back End Data Base Machine", Proc. Inter. Workshop on Database Machines, San Diego, 1982.

[Ba2]

F. Bancilhon & all, "Les V-Relations : Definitions, Modifications, Interrogation", Tech. Notes VERSO 1, 1982.

[Bi]

N. Bidoit, "Un Modele de Donnees Relationel Non Normalise : Algebre et Interpretation", PhD. Thesis, Orsay University, Paris South, 1984.

[BRS]

F. Bancilhon, P. Richard, M. Scholl, "On Line Processing of Compacted Relations", Proc. Inter. Conf. on VLDB, Mexico, 1982, pp 263-269.

[Co]

E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", CACM 13,N°6, 1970, pp377-387.

[D]

C. Delobel, "Normalization and Hierarchical Dependencies in the Relational Data Model", ACM Trans. on Database Systems, N°3, 1978, pp 201-222.

[FMU]

R. Fagin, A. Mendelzon, J. Ullman, "A Simplified Universal Relation Assumption and its Properties", Trans. on Database Systems, N°3, 1982, pp 343-360.

[FT]

P. C. Fisher, S. J. Thomas, "Operations for Non-First-Normal Form Relations", Proc IEEE COMPSAC, 1983, pp 464-475.

[FK]

R. Furtado, L. Kerschberg, "An Algebra of Quotient Relations", Proc. ACM Sigmod Conf., Toronto, 1977, pp 1-8.

[HY]

R. Hull, C. K. Yap, "The Format Model : A Theory of Database Organization", JACM 31 : 2, April 84, pp 210-226.

[IMS]

Information Management System/360, Version 2, General Information Manual, IBM form # GH20-0765.

[JS]

G. Jaeshke, H. J. Scheck, "Remarks on the Algebra of Non First Normal Form Relations", Proc. ACM SIGACT-SIGMOD, PODS Los Angeles, 1982, pp 124-138.

[K]

I. Kobayashi, "An Overview of the Database Management Technology", Tech. Report TRCS-4-1, Sanno College, Kanagawa 259-11, Japan, 1980.

[KTT]

Y. Kambayashi, K. Tanaka, K. Takeba, "Synthesis of Unormalized Relations Incorporating more Meaning", Information Sciences 29, 1983, pp 201-247.

[Mac]

I. A. Macleod, "A Model for Integrated Information Sytems", Proc. Inter. Conf. on VLDB, Florence, 1983, pp 280-289.

[Mai]

D. Maier, "The Theory of Relational Databases", Computer Science Press, 1983.

[MW]

D. Maier, D. Warren, "Specifying Connections for a Universal Relation Scheme Database", Proc. SIGMOD, 1982, pp 1-7.

[Mak]

A. Makinouchi, "A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model", Proc. Inter. Conf. on VLDB, Tokyo, 1977, pp 447-453.

[P]

P. Pauthe,"EVER, un editeur pour V-relations", Thèse de Troisième cycle, Université d'Orsay, 1985.

[SP]

H-J. Scheck, P. Pistor, "Data Structures for an Integrated Data Base Management and Information Retrieval System", Proc. Inter. Conf. on VLDB, Mexico, 1982, pp 197-207.

[SS]

H-J Scheck, M. H. Scholl, "An Algebra for the Relational Model with Relation Valued Attributes", Draft, 1984.

[U]

J. D. Ullman, "Principles of Database Systems", 2nd Edition, Computer Science Press, 1982.

[V]

A. Verroust, "Characterization of Well-Behaved Database Schematas and their Update Semantics", Proc. Inter. Conf. on VLDB, Florence, 1983, pp 312-321.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique