# OPERATIONS AND THE PROPERTIES ON NON-FIRST-NORMAL-FORM RELATIONAL DATABASES

Hiroshi Arisawa    Kunihiko Moriya    Takao Miura

Yokohama National University
Yokohama, JAPAN

Mitsui Engineering & Shipbuilding
Co., Ltd.  Tokyo, JAPAN

## ABSTRACT

In this paper, non first normal form relations (or NFRs) are discussed. First the authors define composition of tuples to introduce NFR and discuss some properties. Then canonical forms of NFRs are defined using "nest" operations. This is optimal in the sense that every 1NF relation can be always transformed into canonical ones and canonical forms have the desired properties to some extent. Also we shall consider data dependency and its effect to NFRs. Finally we consider some algorithms for updating tuples in NFRs with their complexity.

## 1. INTRODUCTION

Most of the recent works concerned with database systems assume theoretical background of relational data model [1]. One of the reasons is due to its mathematical foundation by which we can logically construct and manipulate information without paying attention to physical representation. However, several problems have been pointed out by some researchers. Among them, the reasonableness of the first normal form (or 1NF) is sometimes discussed because it excludes compound value sets from domains [5]. In advanced application processing, we could take more complicated value sets. Basically compound-value problems come from "data semantics", it should be observed from data-model views.

In this paper, we will not pursuit this problem but extend relational model using compound value domains because of simplicity and of mathematical treatment. Also we shall consider the properties including data dependency. Historically some investigations have discussed about non first normal form relations (or NFRs) [6],[7],[8]. But [6] contains ambiguous definitions of NFRs and fails to take advantages of NFRs. [7] considers NFRs as nested relations which are precisely defined on 1NF. We extend this idea to some "normal" form concept like irreducible form and canonical form. Also we describe their properties and algorithms for updating NFR databases with the complexity. By all these discussion we shall show that NFRs have the potential for usefulness.

## 2. CONSIDERATIONS ON COMPOUND VALUES DOMAINS

When we consider compound value domains, a variety of "compoundness" could be discussed. Let us consider a relation that can contain a set of simple values in each field. Even in this simple case some ambiguity exists. For example, suppose SC[Student, Course] relation which represents a student takes a course. When SC contains a tuple $(s, \{c_1, c_2\})$, this says that student s takes courses $c_1$ and $c_2$, or precisely, two tuples $(s, c_1)$ and $(s, c_2)$ are in SC. In this case the $\{c_1, c_2\}$ has no special meaning.

On the other hand, suppose that CP[Course, Prerequisite] relation describes a course has prerequisite courses, and that CP contains $(c_0, \{c_1, c_2\})$. In this case, CP can contain $(c_0, \{c_1, c_3\})$ for another prerequisite condition of $c_0$. As Prerequisite is defined on power set of Course, we can not split those tuples like above. Moreover, we may have $(c_0, \{\{c_1, c_2\}, \{c_1, c_3\}\})$.

Other examples of compoundness are ordered lists, sentences and even relation-valued domains [8].

All the examples make us consider more precise treatement to NFRs. In subsequents, we start with the first pattern, that is, the case the relation is defined on simple domains, because this is the natural extension of relational model and is useful for practical purpose.

Even in such simple notion, the data in NFR has to be manipulated carefully. Lets show following example.

Let $R_1$, $R_2$ be NFRs, defined on {Student, Course, Club} and {Student, Course, Semester} respectively (Fig. 1).

$R_1$

| Student | Course | Club |
|---|---|---|
| $s_1$ | $c_1$, $c_2$, $c_3$ | $b_1$, $b_2$ |
| $s_2$ | $c_1$, $c_2$, $c_3$ | $b_2$ |
| $s_3$ | $c_1$, $c_2$, $c_3$ | $b_1$ |

$R_2$

| Student | Course | Semester |
|---|---|---|
| $s_1$, $s_2$, $s_3$ | $c_1$, $c_2$ | $t_1$ |
| $s_1$, $s_3$ | $c_3$ | $t_1$ |
| $s_2$ | $c_3$ | $t_2$ |

Fig. 1 Example of NFRs

$R_1$

| Student | Course | Club |
|---|---|---|
| $s_1$ | $c_2$, $c_3$ | $b_1$, $b_2$ |
| $s_2$ | $c_1$, $c_2$, $c_3$ | $b_2$ |
| $s_3$ | $c_1$, $c_2$, $c_3$ | $b_1$ |

$R_2$

| Student | Course | Semester |
|---|---|---|
| $s_2$, $s_3$ | $c_1$, $c_2$ | $t_1$ |
| $s_1$ | $c_2$ | $t_1$ |
| $s_1$, $s_3$ | $c_3$ | $t_1$ |
| $s_2$ | $c_3$ | $t_2$ |

Fig. 2 Updated NFRs

$R_1$ contains tuple (s, c, b) when a student "s" takes a course "c" and belongs to a club "b". $R_2$ contains tuple (s, c, t) when a student "s" takes a course "c" in the semester "t". Here, assume a student "$s_1$" stops taking a course "$c_1$". We want to drop the tuples like ($s_1$, $c_1$, *) from both $R_1$ and $R_2$. This corresponds to removing the value $c_1$ of the first tuple in $R_1$, and to removing the first tuple in $R_2$ and adding ({$s_2$, $s_3$}, {$c_1$, $c_2$}, $t_1$) and ($s_1$, $c_2$, $t_1$) to $R_2$ (see Fig. 2). The reasons why these complicated operations broke out in $R_2$ are that we have a Multivalued Dependency (MVD) [2]

Student ->-> Course | Club

in $R_1$, but no MVD in $R_2$.

From the viewpoint of data modelling, it may be explained that each tuple in $R_1$ represents a student entity, and "course" and "club" are its attributes. Therefore $R_1$ represents an entity relation [3]. On the other hand, $R_2$ shows the relationship relation between student's entities and course's entities. We believe there is no distinction between two types of relations taking NFRs into consideration. That is, when we consider compound value domains, we should not assume some dependencies already exist.

Although NFRs have rather complicated structure than 1NFs like the above discussion, the authors claim that the NFR has some "better" properties compared with 1NF. One is the fact that NFRs are much more powerful not only as user view but also as internal view. In practice, the reduction of the number of tuples will contribute to the reduction of logical search space. We call this level of view as realization view.

NFR may have much less tuples than 1NF by putting a group of tuples into one by means of composition. Also NFR may throw away 4NF concept, or it may take advantages of FDs as [7] says.

On the other hand, compound value domains bring some problems into designing. That is, how can we obtain "good" NFRs, how can we keep desired properties at updating NFRs and so on. Subsequently, we'll show the general way to get NFR from 1NF and make clear the properties on NFRs in more detail.

## 3. PROPERTIES OF NON FIRST NORMAL FORM RELATIONS

### 3.1 Basic notation

First we define NFR. We use basically the notation in [4], but we denote a "tuple" in a different way. Given a set of simple domains $D_1$, ..., $D_n$, an orderd set ($e_1$, ..., $e_n$) such that each $e_i$ is in $D_i$ was called an "n-tuple" in the n-ary relation. We denote this tuple as

$$[D_1(e_1) \ldots D_n(e_n)].$$

Now let us extend the "relation" concept to NFR. Given a set of simple domains (or sets of atomic elements) $D_1$, ..., $D_n$, R is said to be non first normal form relation (or NFR) over $D_1$, ..., $D_n$ if and only if R is a set of tuples

$$[D_1(e_{11}, \ldots, e_{1m_1}) \ldots D_n(e_{n1}, \ldots, e_{nm_n})]$$

where $e_{ij}$ belongs to $D_i$. By expanding each tuple component $D_i(e_i)$ to $D(e_{i1}, \ldots, e_{im_i})$, each NFR tuple can represent all the tuples whose domain values are taken from the specified set of values. That is, the above NFR tuple means the set of tuples

$\{[D_1(e_1) \dots D_n(e_n)] \mid e_i \varepsilon \{e_{i1} \dots e_{im_i}\}\}$.

For example, $[A(a_1, a_2)\ B(b_1)]$ means the set of two tuples $[A(a_1)\ B(b_1)]$ and $[A(a_2)\ B(b_1)]$.

Hereafter we denote a relation R as an NFR unless otherwise stated.

## 3.2 Composition of Tuples and Irreducible Forms

Let us define composition and decomposition of tuples for the purpose of getting NFRs. Basic idea of the composing rule was firstly proposed in [12], and some remarkable extentions have been done by Jaeschke and Schek [7].

**Definition 1**
Let r and s be tuples in a relation R such that

$r = [E_1(e_{11}, \dots, e_{1r_1}) \dots E_n(e_{n1}, \dots, e_{nr_n})]$
and
$s = [E_1(d_{11}, \dots, d_{1s_1}) \dots E_n(d_{n1}, \dots, d_{ns_n})]$.

If, for each $i = 1, \dots, n$, $(e_{i1}, \dots, e_{ir_i})$ is set-theoretically equivalent to $(d_{i1}, \dots, d_{is_i})$ except $i=c$, then the operation to create a new tuple

$$[E_1(e_{11}, \dots, e_{1r_1}) \dots E_c(e_{c1}, \dots, e_{cr_c}, d_{c1}, \dots, d_{cs_c}) \dots E_n(e_{n1}, \dots, e_{nr_n})]$$

is called a composition of r and s over $E_c$. This is denoted by $v_{E_c}(r, s)$.

For example, the result of $v_B$ operation on two tuples
$t_1 = [A(a_1, a_2)\ B(b_1, b_2)\ C(c_1)]$
and
$t_2 = [A(a_1, a_2)\ B(b_3)\ C(c_1)]$
is
$t_3 = [A(a_1, a_2)\ B(b_1, b_2, b_3)\ C(c_1)]$.

Composition corresponds to the transformation from 1NF to NFR, because it cannot lose or add any information. That is, this is the syntactical rule by which we have the same amount of information and less tuples.

As composition preserve equivalence between 1NF and NFR, we can define a decomposition which is the reverse operation of composition. However, the result of decompositions depends on the sequence of spliting domain values on $E_c$. We define it in a more restricted way.

**Definition 2**
Let t be a tuple in a relation R
$[E_1(e_{11}, \dots, e_{1t_1}) \dots E_d(e_{d1}, \dots, e_{dt_d}, e_x) \dots E_n(e_{n1}, \dots, e_{nt_n})]$.

The operation getting two tuples
$t_r = [E_1(e_{11}, \dots, e_{1t_1}) \dots E_d(e_{d1}, \dots, e_{dt_d}) \dots E_n(e_{n1}, \dots, e_{nt_n})]$ and

$t_e = [E_1(e_{11}, \dots, e_{1t_1}) \dots E_d(e_x) \dots E_n(e_{n1}, \dots, e_{nt_n})]$

is called a decomposition on $E_d(e_x)$, denoted by $u_{E_d(e_x)}(t)$.

Using the above example, we have $t_1$ and $t_2$ by $u_{B(b_3)}(t_3)$, and we also have other two tuples

$[A(a_1)\ B(b_1, b_2, b_3)\ C(c_1)]$
and
$[A(a_2)\ B(b_1, b_2, b_3)\ C(c_1)]$
by $u_{A(a_1)}(t_3)$.

Both composition and decomposition are defined syntactically depending upon only tuples. In this paper, we restrict ourself to NFR which can be derived from 1NF using composition and decomposion.

Given NFR R we denote its original 1NF relation as $R^*$. Of course $R^*$ has no duplicate tuple and so has R.

**Theorem 1**
Given NFR R, there exists one and only one $R^*$.
(proof) by definition 1 and 2. □

On the other hand, as a 1NF can have several NFRs, we try to find minimal ones in some sense.

**Definition 3**
Let us define irreducible relation. After applying a sequence of compositions, if no more composition is possible without decomposing and re-composing, then the result relation is called an irreducible form relation or just irreducible.

**Example 1**
Thinking about a relation R over A, B, let
$r_1 = [A(a_1)\ B(b_1)]$
$r_2 = [A(a_2)\ B(b_1)]$
$r_3 = [A(a_2)\ B(b_2)]$
$r_4 = [A(a_3)\ B(b_2)]$
be tuples in R.
Applying compositions over A, i.e. $v_A(r_1, r_2)$ and $v_A(r_3, r_4)$, we get an irreducible form relation $R_1$ which contains two tuples
$[A(a_1, a_2), B(b_1)]$ and
$[A(a_2, a_3)\ B(b_2)]$.
Also we can obtain another irreducible form relation $R_2$ containing three tuples
$[A(a_1)\ B(b_1)]$,
$[A(a_2)\ B(b_1, b_2)]$ and
$[A(a_3)\ b(b_2)]$
by $v_B(r_2, r_3)$. □

Above example shows that there could be more than one irreducible form relations derived from 1NF. Clearly, in an irreducible form, the number of tuples is minimal in a sense though it may not be minimum.

## 3.3 Nest Operation and Canonical Forms

Here let us introduce canonical forms of irreducible NFRs using nest operations. [7] discusses the nest operation and its properties.

**Definition 4**
Let R be a relation on domains $E_1, \ldots, E_n$.
Nest operation on $E_i$, denoted by $\nu_{E_i}$ is the successive compositions over $E_i$ as many as possible. The result relation is called a nested relation over $E_i$, denoted by $\nu_{E_i}(R)$. $\nu_{E_i}(\nu_{E_j}(R))$ is abbreviated by $\nu_{E_iE_j}(R)$.

We define canonical forms using the "nest" concept.

**Definition 5**
Let P be a permutation on $E_1, \ldots, E_n$, that is, the sequence $E_1 \ldots E_n$ is replaced by $P(E_1) \ldots P(E_n)$ after applying the permutaion P. The successive nest operations
$\nu_{P(E_1) \ldots P(E_n)}(R)$

is denoted by $\nu_P(R)$. Then it is easy to show that $\nu_P(R)$ is irreducible. Note that we transformed R into $\nu_P(R)$ syntactically and it's always possible. $\nu_P(R)$ is said to be in a canonical form. We have n! permutations and so do canonical forms.

There can exist an irreducible form relation which is not canonical but has fewer tuples than any canonical form relation as following example.

**Example 2**
When a relation $R_3$ over A, B, C has 6 tuples like
$r_1 = [A(a_1)\ B(b_1)\ C(c_2)]$
$r_2 = [A(a_1)\ B(b_2)\ C(c_1)]$
$r_3 = [A(a_1)\ B(b_2)\ C(c_2)]$
$r_4 = [A(a_2)\ B(b_1)\ C(c_1)]$
$r_5 = [A(a_2)\ B(b_1)\ C(c_2)]$
$r_6 = [A(a_2)\ B(b_2)\ C(c_1)]$.
Considering tuples carefully, we have an irreducible form relation $R_4$ which contains three tuples
$[A(a_1)\ B(b_1, b_2)\ C(c_2)]$,
$[A(a_2)\ B(b_1)\ C(c_1, c_2)]$ and
$[A(a_1, a_2)\ B(b_2)\ C(c_1)]$.
But $R_4$ cannot be derived using nest operations. For example, after applying the operation $\nu_{CBA}(R_3)$, we have canonical form relation $R_5$ like
$[A(a_1, a_2)\ B(b_1)\ C(c_2)]$,
$[A(a_1, a_2)\ B(b_2)\ C(c_1)]$,
$[A(a_1)\ B(b_2)\ C(c_2)]$ and
$[A(a_2)\ B(b_1)\ C(c_1)]$.
Thinking over the symmetricity of $R_3$, every canonical form contains 4 tuples. □

Nevertheless, canonical form seems to be "better" than other irreducible forms in the sense that we can syntactically reduce every 1NF to canonical one and that we have a unique NFR which depends only upon a permutation P as in Theorem 2.

**Theorem 2**
Let R be a relation over $U=\{E_1, \ldots, E_n\}$. And let P be a permutation over U.
Then a canonical form relation as a result of $\nu_P$ is unique, that is, the final form is independent of the sequence in composition of tuple-pairs in each $\nu_{E_i}$ operation.

(proof) It's easy because each nest operation $\nu_{E_i}$ preserves the uniqueness property by definition 4. The detailed proof is left to the reader. □

## 3.4 Canonical Form based on FDs and MVDs

Having a canonical form, we have to decide the permutations P. The "best" permutations may stand on the properties which have been investigating in the relational model.

We discuss the strategy to get canonical forms in terms of FDs and MVDs. In this section, we suppose all the relations are in 3NF, which are mechanically obtained [13]. For this purpose, let us define the basic notations.

**Definition 6**
Let R be a relation over $E_1, \ldots, E_n$.
For any e in $E_i$

(1) If e appears in at most one tuple and the tuple has a form [ ... $E_i(e)$ ... ] then we denote it as $E_i{:}R = 1{:}1$,

(2) if e appears in at most one tuple and the tuple has a form [ ... $E_i($ ..., e, ... ) ... ] then we denote it as $E_i{:}R = n{:}1$,

(3) if e appears in more than one tuples and the tuples have a form [ ... $E_i(e)$ ... ] then we denote it as $E_i{:}R = 1{:}n$,

(4) if e appears in more than one tuples and the tuples have a form [ ... $E_i($ ..., e, ... ) ... ] then we denote it as $E_i{:}R = m{:}n$.

Essentially it says the cardinality correspondence between domain values and tuples.

Next, we define "fixedness" concept corresponding to "key" notion on NFR.

**Definition 7**
Let R be a relation over $F_1, \ldots, F_k, E_1, \ldots, E_m$. If, for each $f_1, \ldots, f_k$ where each $f_i$ is in $F_i$, there exists in R at most one tuple which contains all of $f_1, \ldots, f_k$ as a part, then it's said that R is fixed on $F_1, \ldots, F_k$.

In Example 1, R is not fixed on any domain. However, $R_1$ is fixed on A and $R_2$ on B.

Now let us consider FD and MVD with respect to NFR.

**Theorem 3**
Let R be a relation over a set of domains U. Assume FD $F_1, \ldots, F_k \rightarrow E_1, \ldots, E_m$ holds where each $F_i$, $E_j$ is in U. Then any irreducible form relation $R'$ derived from R is fixed on $F_1, \ldots, F_k$ and $E_i$:R' = 1:n for each i=1, ..., m.
(proof) Clearly the FD also holds in $R^*$, and $R^*$ is fixed on $F_1, \ldots, F_k$. Applying all the possible compositions to have the irreducible form, it's sufficient to show the NFR is still fixed on $F_1, \ldots, F_k$. Assume otherwise. Then there should exist the composition which was applied to two tuples whose values on some attribute $E_i$ are different. But it contradicts the property of fixedness on $F_1, \ldots, F_k$. □

**Theorem 4**
Let R be same in theorem 3. Assume MVD
$F_1, \ldots, F_k \rightarrow\rightarrow E_1 | \ldots | E_m$
exists.
Then there exists an irreducible form relation R' which is fixed on $F_1, \ldots, F_k$ and $E_i$:R' = m:n for each i=1, ..., m.
(proof) Similar to theorem 3, and left to the reader. □

Note theorem 4 shows that there may exist an irreducible form which is not fixed on $F_1, \ldots, F_k$ in the case of MVD, as the following example says.

**Example 3**
A relation $R_6$ over A, B, C has 4 tuples and MVD A ->-> B|C is assumed.
$r_1$ = [A($a_1$) B($b_1$) C($c_1$)]
$r_2$ = [A($a_1$) B($b_2$) C($c_1$)]
$r_3$ = [A($a_2$) B($b_1$) C($c_1$)]
$r_4$ = [A($a_2$) B($b_1$) C($c_2$)]
We have an irreducible form relation $R_7$ which contains
[A($a_1$) B($b_1$,$b_2$) C($c_1$)] and
[A($a_2$) B($b_1$) C($c_1$, $c_2$)].
Also we can obtain an irreducible form relation $R_8$ which contains
[A($a_1$, $a_2$) B($b_1$) C($c_1$)],
[A($a_1$) B($b_2$) C($c_1$)] and
[A($a_2$) B($b_1$) C($c_2$)].
$R_7$ is fixed on A, however $R_8$ is not so. □
Moreover, we can show the following theorem.

**Theorem 5**
Let P be a permutaion of U = $E_1, \ldots, E_n$ on which 1NF relation R is defined. Then there exists a fixed canonical form relation where the fixedness is established on at most n-1 domains.
(proof) We will outline the proof and leave the detail to reader. Let R be the NFR and $E_1, \ldots, E_n$ be the nesting sequence. When R is already irreducible, R is fixed on U-$E_i$ for each i. If not, $V_{E_i}(R)$ is fixed on U-$E_i$ for each i. Applying the successive nest operations, the result NFR still holds the fixedness which has been previously established. □

In short, in NFR R, given FD $F_1, \ldots, F_k \rightarrow E_1, \ldots, E_m$ or given MVD $F_1, \ldots, F_k \rightarrow\rightarrow E_1 | \ldots | E_m$, there can exist P by which $V_P(R)$ is canonical and fixed on $F_1, \ldots, F_k$ where P is a permutation of $F_1, \ldots, F_k$. That is, nesting on leftside attributes of FDs or MVDs allows us to get to "better" NFR. The relationships among canonical, fixed and irreducible NFRs are summarized as shown in Fig. 3.
We will show further discussion elsewhere.

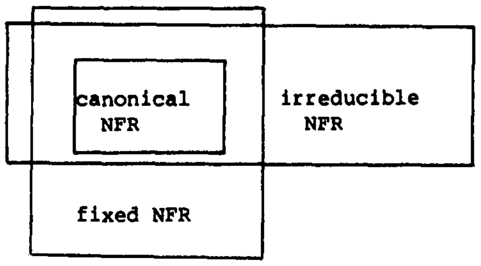

Fig. 3 Relationships among canonical, fixed and irreducible NFRs

## 4. INSERTION AND DELETION OF TUPLES ON NON FIRST NORMAL FORM RELATIONS

As we said, possibly NFR-based database scheme has much less number of relations, in which the number of tuples in each NFR is also drastically reduced. It's certainly one of advantages of NFR compared with 1NF. On the contrary, there are some problems about NFR. First, there might be more than one NFR to represent the amount of information, though 1NF relations give us just one way to do that. Also it's hard to find the "minimum" NFR. Neverthless, theorem 5 shows us there exists one and only one canonical form relation using nest operations.

Another problem is the update of NFR. In 1NF relations update could be applied on a tuple itself, but not in NFR because several tuples may be combined together into one.

Therefore, update operations get more complicated and some might say actual updates happen all over the database. We will show it is not true. When we have the efficient algorithms, NFRs could become useful not only in conceptual level but also in physical representation. Let us move on to the update problem. Remember that R is generally in NFR and $R^*$ its corresponding 1NF relation.

### 4.1 Update Problem on Non First Normal Form Relations

The update problem asks whether there exists an algorithm which is applied to not $R^*$ but R when inserting or deleting a tuple t on R corresponding to $R^*$. Moreover, it's essential that the compexity of the algorithm does not depend on the number of tuples in R but the order of at most $e^n$ where n is the degree.

Now, we show the solution of this problem. (Note in Appendix we describe the theoretical background and the complexity about it.)

Let us define basic concept and functions which are used in the algorithm.

- $\pi(r, E_k)$ : gives the $E_k$-component of tuple r.
- unnest($E_i(e_i)$, t, $t_e$, $t_r$) : gives tuple $t_e$ and $t_r$ which are obtained by the decomposition $u_{E_j(e_j)}(t)$ according to Def. 2.
- compo(x, t, w) : gives tuple w which is obtained by the composition with tuple x and t.
- candt(t, $t_c$, m) : gives a candidate tuple $t_c$ and the minimum value m for given tuple t.
- searcht(t, q) : gives a tuple q in NFR which contains a simple tuple t to be added.
- deletet(q) : delete a tuple q.
- candidate tuples : given tuple t, a tuple s in R is called the candidate tuple of t if and only if one of original simple tuples of s in $R^*$ can be composed with t on $E_i$ and no other tuple in R does not hold this property on $E_j$ for any j<i. Note there exists at most one candidate tuple of t in R (lemma A-1).

## 4.2 Strategy of Insertion Algorithm

Let r=[$E_1(e_1)$, ..., $E_n(e_n)$] be a tuple to be added, and P be $E_nE_{n-1}\ldots E_1$, a permutation.

In order to obtain the same relation of $V_P(R^*+r)$ finally, we have to find the candidate tuple in R of r which is composed with r. Then the candidate tuple may be decomposed, and we have new tuples. After that, we may apply the same operations about new tuples. Moreover so are the tuples which are obtained by composition with tuples to be added (or obtained).

Now we show procedure "insertion" for adding a new tuple to R.

Procedure insertion

```
procedure insertion
 var t:tuple /* for insert tuple */
 begin t := r ;
  recons(t)
 end.
```

Essentially the main operation is the procedure "recons". The procedure "recons" plays role as follows:

Given tuple t, it selects the candidate tuple p by "candt" . Then it executes "unnest" until t becomes composable with the new tuple related to p. Lemma A-2 says this is always possible. But as we have the remaining tuples which are not related to the composition with t, "recons" is invoked recursively to them. After composing t, the composed tuple t' could be composable with other tuples. So "recons" is called again. Note if there exist candidate tuples with respect to t', they are always composable (lemma A-3).

Procedure recons

```
procedure recons (t : tuple)
 var p:tuple /* candidate tuple */
 var $p_e$:tuple /* tuple to be composed with t */
 var $p_r$:tuple /* new tuple of decomposing p */
 var w:tuple /* composed tuple with t and p */
 var j:integer /* index for decomposing order */
 var m:integer /* attrib. number of cand. tuple */
 begin
  candt(t, q, m) ;
  if p <> null then
   begin
    j := n ;
    while j > m do
     begin
      unnest($E_j(e_j)$, p, $p_e$, $p_r$) ;
      if $p_r$ <> null then recons($p_r$) ;
      p := $p_e$ ;
      j := j - 1
     end
    compo(p, t, w) ;
    recons(w)
   end
 end.
```

## 4.3 Deletion Algorithm

Assume the same notation in 4.2. Let us show deletion algorithm. First we have to find a tuple q in R which contains in r by searcht(t, q). Second we apply the operation "unnest($E_i(e_i)$, q, $q_e$, $q_r$)" for i=n to 1 until r=$q_e$. Again, we may have new tuples for each i. For this purpose, the relation should be reconstructed using the algorithm of "recons" in 4.2. Finally, when r=$q_e$, tuple r can be deleted by "deletet".

Now we show the procedure "deletion".

Procedure deletion

```
procedure deletion
 var i:integer /* index for decomposing order */
 var q:tuple /* tuple contains simple tuple t */
 var $q_e$:tuple /* obtained by unnest of q */
 var $q_r$:tuple /* obtained by unnest of q */
 begin   i := n ; searcht(r, q) ;
  while q <> r do
   begin
    unnest($E_i(e_i)$, q, $q_e$, $q_r$) ;
    recons($q_r$) ;
    q := $q_e$ ;
    i := i - 1
   end
```

```
    deletet(q)
  end.
```

## 5. CONCLUSION

We proposed NFR and discussed its properties and the update algorithms on it. We didn't address the data manipulation language which we will show elsewhere. NFR allows database users to take away such decompositions of schema that are forced to occur MVDs, and to discard join operations which originate from the decomposition. In the implementation, it gives us the theoretical foundation enough to reduce the search space in databases.

Although the update algorithms seem to be more complicated than 1NF, the number of composition to keep NFR canonical doesn't depend on the number of tuples. We didn't mean to optimize the algorithm, but the optimization strategy is another problem.

In order to take advantages of NFR, it's necessary to discuss "relations" or predicates in the mathematical meaning that we can find in the recent development of universal relations [10]. That is, NFR stems from the deep consideration of data model itself. It will be necessary to find more fundamental objects of databases.

## References

[1] E.F. Codd : A Relational model of data for large shared databanks, CACM 13-6, pp. 337-387 (1970).

[2] R. Fagin : Multivalued dependencies and a new normal form for relational databases, ACM-TODS, Vol. 2, No. 3, pp. 262-278 (1977).

[3] P.P. Chen : The Entity-Relationship model - toward a unified view of data, ACM-TODS, Vol. 1, No. 1, pp. 9-36 (1976).

[4] J. Ullmann : Priciples of database systems, Computer Science Press (1980).

[5] I. Kobayashi : An overview of the database management technology, tech. report TRCS 4-1, Sanno College (1980).

[6] A. Makinouchi : A consideration on normal form of not-necessarilly-normal data model, 3rd VLDB, pp. 447-453. (1977).

[7] G. Jaeschke, H.-J. Schek : Remarks on the algebra of non first normal form relations, Proc. 1st Principles of Database Systems, ACM, pp. 124-138 (1982).

[8] H.-J. Schek, P. Pistor : Data structure for an integrated data base management and information retrieval system, 8th VLDB, (1982).

[9] H. Arisawa, K. Moriya, T. Miura : Uniformity of data description and query formula, submitted elsewhere.

[10] C. Beeri, P. Bernstein, N. Goodman : A sophisticate's introduction to database normalization theory, 4th VLDB, pp. 113-124 (1978).

[11] H. Arisawa : On the complexity of the update problems on non first normal form relations, Bulletin of the faculty of engineering, Vol.33, Yokohama national university (1983).

[12] H. Arisawa : A conceptual design of a database machine based on a new data model, Proc. of international conference on Entity-Relationship approach to system analysis and design, P. Chen (ed.), pp. 597-614 (1979).

[13] P. Bernstein : Synthesizing third normal form relations from functional dependencies, ACM-TODS, Vol. 1, No. 4, pp. 277-298 (1976).

## APPENDIX

We show the theoretical background and the complexity about Update Algorithms in section 4 without proof. We discussed the complexity of the update problems in the sense of the number of compositions, but not of time complexity because the latter depends heavily on physical representation of NFRs.

**Lemma A-1**

There exists at most one candidate tuple of a given tuple t for each $E_i$.

**Lemma A-2**

Let $t_e$, $t_r$ be tuples which are obtained by $u_{E_i(e_i)}(t)$ operation according to Def. 2.

If there exists the candidate tuple $t_c$ in R of $t_e$, then

$\pi(t_r, E_k) \equiv \pi(t_c, E_k)$ for each k=1, ..., i.

**Lemma A-3**

Let $r_c$ be the candidate tuple in R of a given tuple r, and w be a tuple which is composed with r and $r_c$ (or decomposing $r_c$ if necessary).

If there exists the candidate tuple $w_c$ in $R-r_c$, then

$\pi(w, E_k) \equiv \pi(w_c, E_k)$ for each k=1, ..., i.

All these details and proofs are in [11].

**Theorem A-4**

Insertion and deletion algorithm in section 4 have the complexity of at most $O(e^n)$ where n is a degree of NFR. Note the complexity means the number of compositions.

(proof)

We show the sketch and the complete proof is in [11]. We focus on deletion (the case of insertion is similar).

Let $r = [E_1(\mathbf{e}_1), ..., E_n(\mathbf{e}_n)]$ be the tuple in NFR R which contains $t = (t1, ..., t_n)$, the deleted tuple, where $\mathbf{e}_i$ is a set of values on

$E_i$. Also let $e'_i$ be $e_i - \{t_i\}$, $r_i = [E_1(e_1), \ldots, E_i(e'_i), E_{i+1}(t_{i+1}), \ldots, E_n(t_n)]$ be a tuple deducible from r, and $R_j$ be R if $j = n+1$, $V_{E_n \ldots E_j}(R_{j+1} - [E_1(e_1), \ldots, E_{j-1}(e_{j-1}), E_j(t_j), \ldots, E_n(t_n)])$ if $j \leq n$.

Then, for each j, no tuple in $R_{j+1}$ except $[E_1(e_1), \ldots, E_{j-1}(e_{j-1}), E_j(t_j), \ldots, E_n(t_n)]$ is composable on $E_j$ with $r_j$.

Therefore $r_j$ is not composable any longer on $E_j$. Our goal is to show that, by composing $O(e^{n-j+1})$ times on $R_{j+1} \cup \{r_j\}$, we have $R_j$ where $j < n$, since $j = 1$ means the end of deletion.

We show above by induction.

In the case of $j = n$, there exists at most one tuple in $R_n$ which is composable with $r_{n-1}$. In the case of $j < n$, there exists at most one tuple which is composable with $r_j$ on $E_{j+1}$ in $R_{j+1}$ (Essentially this is lemma A-1).

Let $s = [E_1(e_1), \ldots, E_{j-1}(e_{j-1}), E_j(e'_j), E_{j+1}(a), E_{j+2}(t_{j+2}, b_{j+2}), \ldots, E_n(t_n, b_n)]$ be the tuple. Note $a \not\ni t_{j+1}$, $b_i \not\ni t_i$ for $j+2 \leq i \leq n$. The tuple s can be composed with $r_j$ on $E_{j+1}$ and we have to pick out $[E_1(e_1), \ldots, E_{j-1}(e_{j-1}), E_j(e'_j), E_{j+1}(a), E_{j+2}(t_{j+2}), \ldots, E_n(t_n)]$.

So we have $[E_1(e_1), \ldots, E_{j-1}(e_{j-1}), E_j(e'_j), E_{j+1}(a), E_{j+2}(t_{j+2}, b_{j+2}), \ldots, E_i(b_i), E_{i+1}(t_{i+1}), \ldots, E_n(t_n)]$ for $j+2 \leq i \leq n$. Call those tuples $S_{j+2}, \ldots, S_n$. Also we have $[E_1(e_1), \ldots, E_{j-1}(e_{j-1}), E_j(e_j), E_{j+1}(a), E_{j+2}(t_{j+2}), \ldots, E_n(t_n)]$. This is composable with $r_j$ in $R_{j+1}$, and let $S_0$ be the result.

$S_0$ cannot be composable on $E_{j+1}$. When composing $S_0$ on $E_{j+2}$, we can use inductive assumption. That is, we have at most $f(i)+1$ compositions where $j+2 \leq i \leq n$.
And, in total, the maximum composition count is $f(j+2) + \ldots + f(n) + (n-k-1)$.

By the above consideration we can summarize
$f(j) = (n-k) + 2 \times (f(j+2) + \ldots + f(n))$ in maximum,
$f(n) = 0$ and $f(n-1) = 1$. Calculating them we have the result. □