Institut d'Informatique

Rue Grandgagnage, 21
B-5000 Namur
BELGIUM

Entity-generating Schema Transformation for Entity-Relationship Models

J-L Hainaut

RP-91-003 January 1991

10th Entity-Relationship Conference (San Mateo, 1991)

ENTITY-GENERATING SCHEMA TRANSFORMATIONS FOR ENTITY-RELATIONSHIP MODELS

J-L Hainaut
Institut d'Informatique - University of Namur
rue Grandgagnage, 21 - B-5000 Namur (Belgium)

Abstract. The paper is a contribution to the problem of *semantics-preserving schema restructuring*. First, the notion of schema transformation is precisely analysed, its properties of reversibility are stated and an adequate notation is proposed. Then, the paper develops a general family of *entity-generating transformations*, i.e. operations that add/remove entity types in a schema. This generic transformation is proved to be able to generate not only a large number of popular E-R schema transformations, but also some powerful new ones.

Key-words. Schema equivalence, schema transformation, database design.

1. INTRODUCTION

1.1 Schema transformation

Transforming a source DB schema is a process that produces a new DB schema which has some sort of equivalence with the first one. Generally the new schema satisfies some sort of constraints the first one doesn't meet. For instance, decomposing an unnormalized schema or translating an E-R schema into a DBMS-compliant schema are basically **schema transformations** that preserve the semantic contents of the schemas. In addition, the first transformation gives the schema a higher quality while the second one makes it executable. One is mostly interested in *semantics-preserving* transformations, i.e. transformations such that both source and final schemas represent exactly the same real world facts, though with a different syntax. Such transformations give these schemas the property of **semantic equivalence**. Any database instance structured according to one schema can be losslessly converted into a database instance according to the other one, and conversely, hence the name of *reversible transformation*.

1.2 Schema transformations in database modeling

Schema transformation has been at the very core of many works on database modeling. Most works tackling problems such as E-R to DBMS schema translation, view definition, view integration, multi-base, model translation or reverse engineering pose the fundamental problem of schema equivalence. This problem generally appears in two variants : (1) how to prove that two schemas are equivalent (or more generally to what extent are they equivalent) and (2) how to generate schemas equivalent with another one.

(1) There are several ways to **prove the equivalence of two schemas**. One of them consists in deriving from each schema some kind of complete specification set, such as the closure of the underlying dependency sets, and checking their equality [D'ATRI,84] or proving that each schema implies the other one according to some properties such as dependencies [LIEN,82] [JAJODIA,83]. Another way to prove the equivalence of schemas S1 and S2 is to find a sequence of semantics-preserving transformations such that, when applied to S1, the final schema is equal to S2. This approach is suggested in [D'ATRI,84] and [KOBAYASHI,86] and is the basis of several integration strategies such as those derived from [BATINI,83].

(2) **Generating equivalent schemas** is mainly used in top-down design processes in which E-R (or binary, or object-based) schemas are derived, refined, normalized, simplified and translated in DBMS-compliant schemas, through semantics-preserving transformations. Let's only mention some of such proposals : [HAINAUT,81], [D'ATRI,84], [NAVATHE,84], [GIRAUDIN,85], [BERT,85], [REINER,86], [KOZACZYNSKY,87], [MOTRO,87], [ROSENTHAL,88], [LING,89].

Be they explicit or implicit, schema transformations clearly appear as fundamental formal tools in almost all DB modeling activities.

1.3 Schema transformation in the literature

R. Fagin defines *a transformation from one database schema into another* as *a mapping **f** from the valid instances* (e.g. ***s***) *of the first database schema into valid instances* (denoted by ***f(s)***) *of the second*. [FAGIN,81]. Furthermore, he observes that *a lossless transformation is defined by a 1-1 mapping*, such that ***f(s)*** uniquely determines ***s***. J. Rissanen [RISSANEN,77] evokes decomposition transformations which are not only 1-1 but also *onto* valid instances of the second schema, in such a way that **any** instance of the latter schema can be obtained by mapping one valid instance of the first schema with ***f*** . Following Casanova [CASANOVA,84], Rosenthal and Reiner [ROSENTHAL,88] call schemas *content-equivalent* if *there is an invertible mapping between their possible instantiations* Moreover, *an instance mapping is invertible if it is total, surjective and injective [...]*. The concept of schema transformation has long been recognized in the literature as *a mapping between data instances*. However, few authors have analysed it as a *mapping between schemas*.

In the literature, the notion of schema transformation is generally used into the more general framework of DB design, view integration and view derivation. Some works, however, are mainly or exclusively dedicated to schema transformation and give a more in-depth insight into this mechanism [HAINAUT,81], [GIRAUDIN,85], [KOBAYASHI,86], [ROSENTHAL,88]. In many papers, only adhoc transformations are proposed, i.e. transformations that allow the elimination of some undesirable constructs. Moreover they are sometimes presented as intuitive, obvious (therefore undemonstrated) schema restructuring techniques. Finally, they are often proposed for simplified formalisms, such as binary models [HAINAUT,81], [REINER,86], [GIRAUDIN,87], [ROSENTHAL,88] or strongly constrained E-R models [D'ATRI,84].

1.4 Classification of schema transformations

In *entity-based models*[1], the transformations can be classified into two categories, namely the **entity-preserving** transformations, that leave the set of the entity types unchanged, and the **entity-generating** transformations, that add new entity types in the final schema[2]. Entity-preserving transformations are often based on projection/join operations defined on relational-like expression of E-R schemas [D'ATRI,84]. As we shall see, entity-generating transformations need more powerful operators.

Synthetizing the proposals found in the literature, and adding some schema rearrangement techniques from the current practice, lead to a huge number of specific transformations. In the scope of E-R schemas, the author has identified up to 50 different meaningful schema transformations

---

1 Entity-based models offer specific constructs for representing real-world objects independently of their properties. This category includes E-R model variants, the OO models, the NF^2 models that support object identity, DBTG model, binary models. They are opposed to value-based models, such as the standard 1NF relational model.

2 If such a transformation is invertible, it has an inverse transformation which is *entity-removing*.

[HAINAUT,90a]. However, it is possible to prove that, at an adequate level of abstraction, all these techniques can be derived from a very small set of general transformations. [HAINAUT,81] proposes 4 generic transformations covering most needs in model translation. In [D'ATRI,84], the authors claim that all entity-preserving transformations are synthetized into 6 semantics-preserving graph restructuring operators. [KOBAYASHI,86] proposes 4 generic transformations that are claimed to cover most of the useful schema retructuring needs.

1.5 About this paper

This paper is a contribution to the study of schema transformation. Its main objectives are,

- to analyse further the concept of schema transformation, and
- to propose an abstract generic transformation from which a large class of popular (and some less known) semantics-preserving, entity-generating schema transformations can be derived.

Section 2 studies the notion of *schema transformation* in general together with its properties of semantics-preservation or *reversibility*. Section 3 develops the *extension transformation* and some of its major properties independently of the data models. These definitions are first developed into the relational formalism for simplicity and generality. In section 4, they are specialized to entity-based models, and more specifically applied to E-R models. Section 5 develops the generation of some popular E-R schema transformations as specializations of the proposed transformation. In addition, it shows how new powerful transformations can be generated the same way.

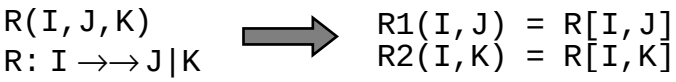
1.6 Prerequisite

The reader is supposed to be somewhat familiar with the relational theory (see for instance [MAIER,83] or [ULLMAN,88]). Let's however recall some important notions and their notation. Let $U = \{A_1, A_2, \ldots, A_N\}$ be a finite set of symbols called *attributes*, each denoting a finite set of values called *domain*. A *relation schema* (R,C) is defined as a named set of attributes (denoted by R(U) or $R(A_1, A_2, \ldots, A_N)$), along with a set of constraints C, such as dependencies and keys. N is called the *degree* of R. The complete expression of R(U) is $R(A_1:D_1, .., A_N:D_N)$, where A_i is the name of an attribute and D_i the name of its domain. In many cases, the specification of the domain will be dropped, specially when the names of the attribute and of its domain are the same. A *relation* r over relation schema (R,C), also called an *instance* of R, is a subset of the cartesian product of the domains denoted by U that satisfies constraints C. A *relational schema* S is a collection of relation schemas, possibly along with inter-relational constraints such as inclusion dependencies. An *instance* s of relational schema S is a collection of relations such that there is one and only one relation for each relation schema of S and that satisfies the inter-relational constraints of S. The following notions will also be used : functional (FD) and multivalued (MVD) dependencies (including FD and MVD holding in non primitive relations, as in $S1*S2: A \longrightarrow B$), inclusion dependency or constraint, project and join operators (including the generalised join, denoted by $*R_i, i \in [1..n]$), decomposition of a relation, key (a key will be underlined), third and Boyce-Codd normal form, data-preserving (lossless) and dependency-preserving decomposition. Finally, we shall often make use of letters I,J,K to denote subsets of U. If $I = \{A_1, .., A_i\}$, the symbol I should read $A_1, .., A_i$, when appearing in a relation schema denotation, so that $R(I,A_j)$ is to be developed into $R(A_1, .., A_i, A_j)$.

2. SCHEMA TRANSFORMATION

2.1 Intuitive analysis of a classical example

The project/join decomposition of a relation schema is a popular schema transformation in the relational theory. In one of its simplest variant it can be specified as follows [FAGIN,77] :



That expression is to be interpreted in the following terms :

"*(1) any relational schema R, whose set of attributes can be partitioned into non-empty subsets I, J and K, and in which multivalued dependencies I →→ J|K hold,*

(2) can be replaced by schema R1 defined on I, J and schema R2 defined on I, K;

(3) for any instance r of R, instance r1 of R1 is obtained by r[I, J] and instance r2 of R2 is obtained by r[I, K]; "

There is an interesting additional property stating that R=R1*R2; this expression states a sort of reversibility property of the decomposition transformation. It is to read :

"(4) joining target instances r1 and r2 obtained in (3) gives back instance r."

This discussion enlights four major parts in the definition of a transformation, namely,

(1) The general description of any source schema candidate for transformation; in addition, that description identifies the relevant components (here, R, I, J and K).

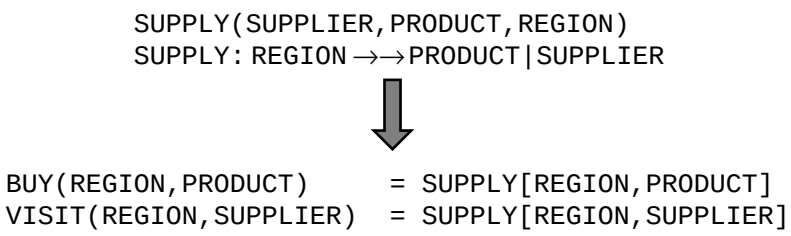
(2) The general description of the target schema; that description identifies the new components, and their relationships with the source components (here, R1 on I, J and R2 on I, K).

(3) The computing rules that define the instances of the target schema from any valid instance of the source schema.

(4) If the transformation has some property of reversibility, the computing rules that would allow restoring the source instances from the computed target instances.

Description (1) and (2) can be thought of as *pattern matching predicates* giving pre- (and post-) conditions that source (and target) schema must (and will) satisfy, and identifying the components involved in the transformation, i.e. giving them generic names. Computing rules are expressed in any adhoc data manipulation or definition language associated with the data model for which the transformation is defined.

The transformation proposed above is a *generic transformation,* in which R, I, J, K, R1 and R2 are generic names, playing the role of *variables* in the predicates. Substituting actual database object names (*values*) for these variables leads to a *specific transformation*. The following expression defines a specific transformation that has been obtained by replacing R with SUPPLY, I with {REGION}, J with {PRODUCT}, K with {SUPPLIER}, R1 with BUY and R2 with VISIT.



2.2 Schema transformation : concepts and notation

Let ***F*** be a formalism for database definition and manipulation including a set ***O*** of primitive operators on database instances (i.e. ***F*** defines a database model). Let $\mathbf{Z}$ be the set of all the schemas that can be defined in ***F***. For each ***S*** of $\mathbf{Z}$, ***inst(S)*** is the set of all the valid database states (or instances) of ***S***.

A *schema transformation* Σ is defined as a pair of mappings <***T,t***> such that,

$$\mathbf{T} : \mathbf{Z} \longrightarrow \mathbf{Z}$$
$$\mathbf{t} : \mathbf{inst(S)} \longrightarrow \mathbf{inst(T(S))}, \quad \forall \mathbf{S} \in \mathbf{Z} \ .$$

Mapping ***T*** tells how to produce the target schema from any source schema, while mapping ***t*** tells how to translate each instance of the source schema into an instance of the target schema.

Generally, a given transformation will apply on some specific schemas only, and more precisely on parts of them that satisfy some preconditions. The parts concerned are best defined by a *pattern matching predicate* ***P*** that identifies the concerned substructures and their components in source schemas. The transformed target schemas can be characterized by a similar predicate ***Q***, acting as a postcondition and stating what that schema looks like, i.e. what are the new components, what are the discarded ones and what are the relationships between them. Moreover, mapping ***t*** can be defined by any combination of operators from ***O*** that produces a valid instance of ***T(S)*** from any valid instance of ***S***. When needed, a transformation <***T,t***> will be given the more explicit notation <***P,Q,t***> as well[3].

Example. Schema 2.1 gives the definition of a simple relational transformation. Predicates ***P*** and ***Q*** are denoted according to one of the natural syntaxes of the relational formalism (***F***). For instance, ***P*** states that there exists a set of domains U, covered by distinct, non-empty attribute subsets I and J, and a relation schema R defined on U. ***Q*** states that applying ***T*** to any schema S such that P(S) consists in dropping R, defining a relation schema R1 on I, a relation schema R2 on J, and defining an equality constraint on projections of R1 and R2 on common attributes. ***t*** states that given any instance r of R, the corresponding instance of R1 (R2) is obtained by projecting r on I (J). The transformation is obviously lossy, since the conditions for safe decomposition are not satisfied.

```
P:  - R(U)
    - I∪J = U; I ≠ J; I,J ≠ {}
Q:  - R1(I), R2(J)
    - R1[I∩J] = R2[I∩J]
t:  - R1 = R[I]
    - R2 = R[J]
```

Schema 2.1 - *A generic schema transformation in the relational formalism.*

Schema 2.2 describes a *specialization* of the generic transformation 2.1. It has been obtained by substituting names of actual relational objects for variables of P and Q, namely U, I, J, R, R1 and R2. We propose to express specializations of transformation <T,t> on schema S by <S, T(S),t>, i.e. by the specification of the source and of the target schemas, together with the instance mapping specialized for S and T(S)[4].

---

[3] For completeness, T and t are identity mappings for schemas or parts of schemas S such that ¬P(S). T is therefore a total mapping.

[4] It is interesting to note that Q and P are made up of two kinds of formulae that behave differently in the substitution process. Formulae of the first kind are *meta-formulae* that define valid substitutions and that disappear in the specific

```
source schema:
    C(CUST#,CUSTNAME,CITY,AMOUNT)
target schema:
    CC(CUST#,CUSTNAME,CITY)
    CA(CITY,AMOUNT)
    CC[CITY] = CA[CITY]
instance mapping:
    CC = C[CUST#,CUSTNAME,CITY]
    CA = C[CITY,AMOUNT]
```

Schema 2.2 *- A specific transformation derived from transformation 2.1 by the following substitutions:* `R=C; U={CUST#,CUSTNAME,CITY,AMOUNT}; I={CUST#,CUSTNAME,CITY}; J={CITY,AMOUNT};` *etc. Let's observe that this substitution is valid since* `I∪J=U; I≠J; I,J≠{}`.

The syntax used for expressing P, Q and t may vary from one formalism to another, and its choice is irrelevant to the notion of transformation developed hereabove. For instance, a graphical representation could be better suited for E-R schema transformations, as will be seen later.
Let's consider two transformations $\Sigma_1 = <P_1,Q_1,t_1>$ and $\Sigma_2 = <P_2,Q_2,t_2>$ such that $Q_1(S) \Rightarrow P_2(S)$, for any S. It is therefore possible to consider a third transformation Σ_3 as the ***composition*** of Σ_1 and Σ_2, defined as:

$$\Sigma_3 = \Sigma_2 \circ \Sigma_1 = <P_1,Q_2,t_2 \circ t_1>$$

In the latter expression, mapping $t_2 \circ t_1$ is defined by $t_2 \circ t_1(s) = t_2(t_1(s))$.

2.3 Reversible and symmetrically reversible transformations

A transformation $\Sigma_1 = <T_1,t_1> = <P_1,Q_1,t_1>$ is ***reversible*** (invertible) or lossless, if one can define another transformation $\Sigma_2 = <P_2,Q_2,t_2>$, called **the inverse of** Σ_1, such that any instance **s** of any schema **S** that satisfies P_1 can be built back by applying t_2 to the transformed instance $t_1(\mathbf{s})$, or more precisely, such that,

$$\forall\ \mathbf{S} \in\ Z, \qquad [\ Q_1(T_1(\mathbf{S})) \Rightarrow P_2(T_1(\mathbf{S}))\ ] \wedge [\forall\ \mathbf{s} \in\ \text{inst}(\mathbf{S}),\ t_2(t_1(\mathbf{s})) = \mathbf{s}\ ]$$

Such a transformation is called an *R-transformation* and is said to have the *R-property*. A generic R-transformation will be denoted by $<P_1,Q_1,t_1,t_2>$ (as far as T_1 is concerned, P_2 and Q_2 are of no interest). When specifying a specific R-transformation, a second instance mapping expression according to t_2 should be added : <S,T(S),t1,t2>. Schema 2.3 shows an interpretation of the traditional MVD-based version of the lossless project/join (P/J for short) decomposition informally used in section 2.1 .

```
P:   R(I,J,K); R:I →→ J|K
Q:   R1(I,J); R2(I,K)
t1:  R1 = R[I,J]; R2 = R[I,K]
t2:  R  = R1*R2
```

---

schema (e.g. `I∪J=U` or `I,J≠{}`". Formulae of the second kind are translated in the substitution process (e.g. `R1[I∩J]=R2[I∩J]`). Discussing this distinction pertains to a general theory of schema transformation that is still to be developed, and is beyond the scope of this paper. [KOBAYASHI,86] discusses some aspects of these formulae but without distinguishing them.

Schema 2.3 *- The traditional P/J decomposition, expressed as an R-transformation.*

An R-transformation $<P_1,Q_1,t_1,t_2>$ is called ***symmetrically reversible*** if its inverse transformation $<P_2,Q_2,t_2>$ is an R-transformation as well (hence denoted by $<P_2,Q_2,t_2,t_1>$). Such a transformation is called an *SR-transformation* and has the *SR-property*. In that case, $<P_2,Q_2,t_2,t_1>$ has the SR-property as well. Moreover, $Q_1 = P_2$ and $Q_2 = P_1$. Therefore the notation $<P,Q,t_1,t_2>$ specifies completely a couple of inverse SR-transformations. When specifying a specific SR-transformation, we must be aware that swapping the source and target schemas and swapping the instance mappings gives the specification of the inverse transformation. Therefore, this specification gives a two-way description. An SR-transformation defines a bijection between inst(S) and inst(T1(S)).

These notions formalize a bit further the concept of semantic equivalence and semantics preservation used by many authors. It is easy to prove that a *1-1* transformation, as defined in [FAGIN,81], is an R-transformation, and a transformation which is both *1-1* and *onto*, is an SR-transformation. [KOBAYASHI,86] defines SR-transformations only, through forward and backward mapping rules. Content-equivalent transformations of [ROSENTHAL,88] are SR-transformations as well.
It can be proved that if Σ_1 and Σ_2 are both R- (or SR-)transformations, then their composition Σ_3 is an R- (or SR-)transformation as well. If Σ_1 and Σ_2 are inverse of each other, $t_2 \circ t_1$ is the *identity* mapping.

The P/J transformation 2.3 is known to be lossless, hence reversible [FAGIN,77]. However, as quoted in [FAGIN,81] and [KOBAYASHI,86], it is clearly not an SR-transformation, sinceits inverse $<Q,P,t_2,t_1>$ would be an R-transformation only if `R1=(R1*R2)[I,J]` and `R2=(R1*R2)[I,K]`, a property that cannot be claimed for arbitrary instances of R1 and R2. The latter condition would be implied by the assertion `R1[I]=R2[I]`. Thus, thanks to a slight modification of the definition of the P/J decomposition, we can be provided with a true SR-transformation defined in Schema 2.4.

```
P:   R(I,J,K); R:I →→ J|K
Q:   R1(I,J); R2(I,K); R1[I]=R2[I]
t1:  R1 = R[I,J]; R2 = R[I,K]
t2:  R = R1*R2
```

Schema 2.4 *-The P/J decomposition revisited in order to give it the SR-property.*

As stated above, the inverse transformation $<Q,P,t_2,t_1>$ has the SR-property. By analogy with its inverse, it could be called a *join/project* transformation. Though it is generally ignored in the traditional relational theory (it is a *denormalization* process), it is largely used in physical DB design. We shall make use of it in the following for demonstration purposes.

3. THE EXTENSION TRANSFORMATION

3.1 Introduction

Most works pertaining to the relational database theory are based on the assumption of a fixed set of domains and attributes. The transformations based on it consist in defining different arrangements of attributes that satisfy given criteria such as data or dependencies preservation, or redundancy minimization. Even some proposals for E-R transformation stick to this principle ([D'ATRI,84] proposes no transformations that would augment the number of entity types). Some recent proposals allow the *extension* of the set of attributes by including new attributes, generally functionally dependent on the previous ones. Four techniques are worth mentioning within the scope of this paper. (1) In the quest for a more powerful normalization process, particularly in case of non-acyclic, MVD-based, schemas, C. Beeri and M. Kiefer [BEERI,86] suggest the introduction of a new attribute, a process they call *schema extension*. (2) In the realm of complex object modeling, S. Abiteboul and C. Beeri [ABITEBOUL,87], [HULL,87] have introduced a specific algebraic operator for constructing *new types* by *extending* a given type with a new component. (3) Translating a relationship type into an entity type is a common practice in E-R schema transformation. This operation consists in extending an E-R schema with a new entity type. [ROSENTHAL,88] is one of the most recent proposition that formalizes that operation. (4) The encode/decode transformations of [KOBAYASHI,86] formalize the introduction of new domains as function application on previously defined domains.
The transformation developed in this paper belongs to this family. In short, the proposed *extension transformation* of relation schema R is based on *extending* R with a new attribute, defined on a new domain, then possibly decomposing R. It is applicable to any schema, whatever its normalisation state. Since the transformation will be proved to have the SR-property, it also proposes conditions under which attributes and domains can be removed from a schema. In the E-R interpretation of the transformation, the new domain will denote an entity type. Therefore, it belongs to the *entity-generating* transformation family.

Though the proposed transformation is not dedicated to a specific database model, a unique formalism is needed to develop, validate and analyse the transformation before applying it to entity-based models. The **standard relational model** will be chosen for that purpose, due to its simplicity and to the availability of well established formal tools. Rules will then be given to apply the results to higher-order models, and particularly to Entity-Relationship models.

3.2 The extension transformation of a relational schema

From an intuitive viewpoint, the *extension transformation* can be sketched as follows. Let PROGRAM(TEACHER,SUBJECT,DATE) be a source relation schema describing facts about *teachers* that teach *subjects* on given *dates*. Suppose that the <TEACHER,SUBJECT> sub-tuples that appear in instances of PROGRAM are now perceived as so significant that we decide they are worth being represented by the new domain LECTURE. We can then propose a revised version of PROGRAM that reads R2(LECTURE,DATE). We also need the companion relation schema R1(LECTURE, TEACHER,SUBJECT) that *defines* each LECTURE value in terms of TEACHER and SUBJECT values. The target schema is made up of R1 and R2, or of further decomposition thereof.

The transformation develops in two steps. The first one introduces the new domain while the second one, which is optional, decomposes the target schema into simpler fragments. According to the needs, we shall use either the first step or the complete transformation.

The extension transformation, step 1 : basic processing

Let S1 be a relational schema consisting of the non empty set of attributes U, with cardinality N, and the relation schema R, in which no (non-trivial) dependencies hold.

$R(U)$ (schema S1)

Let's then consider an arbitrary partition $\{I, J\}$ of U, such that $I=\{A_1, A_2 .. A_n\}$ is not empty ($n > 0$). Schema S1 can be rewritten as schema S1'.

$R(I,J)$ (schema S1')
$I \neq \{\}$

Finally, let X be an arbitrary domain and B an arbitrary bijection from X onto $R[I]$ (i.e., $B: X \longleftrightarrow R[I]$). X is to be interpreted as an arbitrary set of *denotations* for tuples of $R[I]$, B being the *denotation function*. Schema S1", where X and B appear, conveys the same information as schema S1':

$R(I,J)$ (schema S1")
$B(X,I)$
$B: X \longrightarrow I$
$B: I \longrightarrow X$
$B[I]=R[I]$

Applying the *join/project* transformation (section 2.3) leads to schema S2,

$R'(X,I,J) = R*B$ (schema S2)
$R': I \longrightarrow X$
$R': X \longrightarrow I$

According to the SR-property of the *join/project* transformation, $R=R'[I,J]$. The keys of R' are $\{X,J\}$ and $\{I,J\}$, so that R' is in 3NF but not in BCNF if J is not empty. The decomposition $\{R1', R2'\}$ of R' according to $R': X \longrightarrow I$, is in BCNF, hence schema S3. Though $R1' = B$, according to the SR-property of the join/project transformation, we shall keep the projection $R1'$ in the following.

$R1'(X,I)$ (schema S3)
if $n<N$: $R2'(X,J)$
$R1':I \longrightarrow X$
$R1':X \longrightarrow I$
if $n<N$: $R1'[X] = R2'[X]$

The extension transformation, step 2 : post-processing

This first result can be kept as such. However, it can be further decomposed as follows. If I is made up of more than one attribute (i.e. if $n > 1$) the functional dependency ($R1':X \longrightarrow I$) allows the decomposition $\{R1_1', R1_2', \ldots, R1_m'\}$ of $R1'$ according to any partition of I, and particularly to the n attributes $A_1, A_2 .. A_n$ of I. However, this decomposition is no longer dependency-preserving since $R1':I \longrightarrow X$ has been lost. We must define an additional inter-relation constraint that expresses this lost FD. Hence schema S4, obtained by choosing to partition I according to each of its attributes.

$R1_i'(\underline{X},A_i)$, $i \in [1..n]$ (schema S4)
if $n<N$: $R2'(X,J)$
$*R1_i':I \longrightarrow X$
if $n<N$: $R1_i'[X] = R2'[X]$ $i \in [1..n]$

Using the general formalism for transformation description proposed in section 2, the complete *extension transformation* (both steps) can be described as <P,Q,t_1,t_2> as follows,

P:	$R(U)$; $U=\{I,J\}$; $I=\{A_1,A_2,..,A_{\|I\|}\}$; $\|I\|>0$
Q:	$R1_i(\underline{X},A_i)$ $i \in [1..\|I\|]$ $R2(X,J)$ *if* $J \neq \{\}$ $*R1_i:I \longrightarrow X$ $R2[X] = R1_i[X]$ $i \in [1..\|I\|]$ *if* $J \neq \{\}$ X does not appear in any other relation schema
t_1:	choose any arbitrary domain X and relation $B(\underline{X},\underline{I})$ such that $B[I]=R[I]$ $R1_i = (R*B)[X,A_i]$, $i \in [1..\|I\|]$ $R2 = (R*B)[X,J]$, *if* $J \neq \{\}$
t_2:	$R = ((*R1_i)*R2)[I,J]$ $i \in [1..\|I\|]$

Let's rewrite more formally the small introductory example as a specific extension transformation :

```
source schema :
   PROGRAM(TEACHER,SUBJECT,DATE)
target schema :
   R11(LECTURE,TEACHER)
   R12(LECTURE,SUBJECT)
   R2(LECTURE,DATE)
   R11*R12: TEACHER,SUBJECT ⟶ LECTURE
   R11[LECTURE] = R12[LECTURE] = R2[LECTURE]
instance mapping 1 :
   choose  LECTURE and B(LECTURE,TEACHER,SUBJECT) arbitrarily such that
           B[TEACHER,SUBJECT] = PROGRAM[TEACHER,SUBJECT]
   R11 = (PROGRAM*B)[LECTURE,TEACHER]
   R12 = (PROGRAM*B)[LECTURE,SUBJECT]
   R2  = (PROGRAM*B)[LECTURE,DATE]
instance mapping 2 :
   PROGRAM = (R11*R12*R2)[TEACHER,SUBJECT,DATE]
```

3.3 Some properties of the extension transformation

We shall concentrate on five important issues, namely reversibility, the degree of the target schema, preservation of functional dependencies holding in R, the parametrization of the transformation, and the number of schemas that can be generated that way.

The extension transformation is an SR-transformation

The introduction of X and B bears no semantics, and is therefore an SR-transformation. The building of R*B is through the *join/project* transformation (Schema 2.4, inverse transformation) which is SR as well thanks to the hypothesis that B[I]=R[I]. Furthermore, dependencies[5] $X \twoheadrightarrow A_1|A_2|..$

[5] These dependencies are FD and therefore MVD as well, so that the P/J decomposition (Schema 2.4) applies.

$|A_{|I|}|J$ hold in R*B, making it valid to decompose R into $R1_i$,R2. Retaining the properties $R2[X]=R1_i[X]$ on the resulting fragments gives the decomposition the SR-property. Being the composition of SR-transformations the extension transformation is therefore an SR-transformation.

Degree of the resulting schema

In some conditions, the transformation tends to reduce the degree of the source schemas. This property can be useful in physical schema design and in model translation.

The maximum degree of the target schema is $max(2,|U|-|I|+1)$. Therefore, the extension transformation reduces the degree of a schema when $|U|>2$ and $|I|\geq 2$. In addition, the resulting schema is purely binary (degree 2) *iff* $|I|\geq|U|-1$. Transformations where $|I|=1$ do not reduce the degree of the source schema, but are not useless however as will be seen later.

Preservation of functional dependencies

This issue is related to the problem of constraint propagation in schema transformation. Its complexity has been asserted in [KOBAYASHI,86] for example. We shall concentrate on FD only. The development in section 3 is both valid and complete only if no non-trivial FD hold in R[6]. It is still valid but can be incomplete in case of non-trivial FDs. Indeed, these FD can be lost in the project/join decompositions {R1',R2'} and {$R1_1$',$R1_2$', ...}. Furthermore, the FD $R:I \longrightarrow X$ can be lost in the target schema if $|I| > 1$, leading to complex inter-relation FD. A comprehensive study of FD preservation in the extension transformation is beyond the scope of this paper. However, some useful observations can be done for BCNF schemas.

The *no non-trivial FD* hypothesis states that R is at least in BCNF and that U is the only key of R[7]. In BCNF schemas, two other cases only may occur, namely schemas with one non-key attribute (a schema with several non-key attributes can be decomposed according to the key dependencies), and schemas with several keys. It can be proved that all the FD of R are preserved in the following cases :

- $|I| = 1$ ($I \longrightarrow X$ is preserved as well)
- $|I| = |U| = 2$ (moreover, if $A_1 \longrightarrow A_2$, then $I \longrightarrow X$ is preserved as well)
- if K is a key, all the FD of which K is the determinant are preserved if $I \subseteq K$ or $I \subseteq U-K$

Loss of a specific FD ***f*** can be avoided by restricting the extension transformation as follows :

- limiting the {I,J} partition to patterns that preserve ***f***;
- limiting the decomposition of R1'(X,I) in schema S3 in such ways that ***f*** is preserved in a non relation schema $R1_i$'.

In the other cases, the lost FD must be expressed through complex inter-relation expressions as exhibited in section 3.2, Schema S4. This problem is analysed to some extent in [KOBOYASHI,86] for three kinds of constraints, including FD. However, the paper gives no help as to when FD are preserved and what are the FD-preserving transformations.

Parametrization of the transformation

Specializing the extension transformation to an actual schema assigned to R(U) leaves some generic variables of predicate P still unresolved, and some further decisions to be done. We can think of these

---

6 A trivial FD is of the form K $\longrightarrow$ K', where K' $\subseteq$ K.

7 This hypothesis may seem very strong. It should be kept in mind that our ultimate goal is to study schema transformations in entity-based models, and not in relational models in general.

choices as a sort of *parameter setting* for the transformation of R(U). The choices can be synthetized as follows :

- assign a value to I (Schema S1');
- one-step or two-step transformation (Schema S3 or S4)
- in case of post-processing, choose a partition for I (Schema S4)

According to these choices, an actual source schema can be transformed into different target schemas.

Number of derived schemas

For any schema R(U) with degree N, with no non-trivial FD, it can be proved that the number E of schemas that are equivalent to R, including R itself, according to the two-step extension transformation applied recursively (limited to cases where $|I| > 1$), is obtained by the following formula[8],

$$E(1) = 1$$

$$E(N) = 1 + \sum_{k=2}^{N} C_N^k \times E(N+1-k)$$

For instance, a source *3-ary* relation schema will produce a set of 8 equivalent schemas, while a source *4-ary* schema will be in a set of 58 equivalent schemas.

4. THE EXTENSION TRANSFORMATION IN ENTITY-BASED MODELS

Due mainly to the inherent semantic complexity of entity-based models their formalization has not been paid as much attention as that of the relational model, or at least has not led to stabilized results yet. However, the results of the relational theory can be applied to entity-based models provided that one can find a complete set of mapping rules of the relational model onto these models (see [D'ATRI,84] for instance). The Generic Entity-Relationship (GER) model proposed in [HAINAUT,89] and [HAINAUT,90b] is an extended relational model that gives a relational interpretation of most entity-based models, and that can be mapped onto them. We shall summarize the main concepts and mapping rules of the GER model that are of interest in this paper through some examples, then generalize the extension transformation to entity-based models. We shall choose the E-R model as the most representative of them.

4.1 The GER model : a relational interpretation of entity-based models

The GER model is mainly a *reference model* for entity-based models. It allows the expression of entity-based structures by means of an **extended relational formalism**. A short presentation of the GER model can be found in appendix. We will only illustrate three of the constructs we will use in the following :

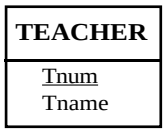
(1) representing an **entity type** and **its attributes** : schema 4.1;

(2) representing a **relationship type** : schema 4.2;

(3) representing a **functional relationship type** (i.e. binary and *many-to-one*): schema 4.3.

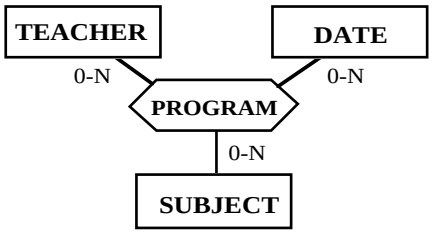
---

8 The binomial factor is in European notation where $k \leq N$.



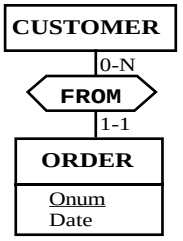

```
TEACHER : entities
d_of_T(TEACHER,Tnum:integer,Tname:char 20)
```

Schema 4.1 *- Graphical (left) and GER (right) expressions of a simple entity type. The GER schema declares entity type* `TEACHER` *as a domain which is a subset of the universal domain entities. The relation schema* `d_of_T` *describes entity type* `TEACHER` *by associating with it its attributes* `Tnum` *and* `Tname` *and their domains. Note that* `d_of_T` *has two keys, namely* `TEACHER` *and* `Tnum` *(from now on, we shall drop the specification of attribute domains whenever no ambiguity may arise). Examples of optional, compound and multivalued attributes will be proposed later on.*




```
TEACHER: entities
SUBJECT: entities
DATE   : entities
PROGRAM(TEACHER,SUBJECT,DATE)
```

Schema 4.2 *- Graphical and GER expressions of an E-R schema consisting of three entity types and one 3-ary relationship type*[9]*. Should relationship type* `PROGRAM` *have been given attributes, they would have been added to the GER relation schema* `PROGRAM` *as well.*




```
CUSTOMER : entities
ORDER    : entities
d_of_O(ORDER,Onum,Date,CUSTOMER)
d_of_O[ORDER] = ORDER
```

Schema 4.3 *- Functional (binary, many-to-one, without attributes) relationship types are represented as in Schema 4.2. However, an alternate concise GER representation, obtained by joining its expression with that of the entity type attributes description, can sometimes be prefered to express complex integrity constraints and transformations. Note the inclusion (equality) constraint that states that each* `ORDER` *entity must appear in at least one* `d_of_O` *tuple.*

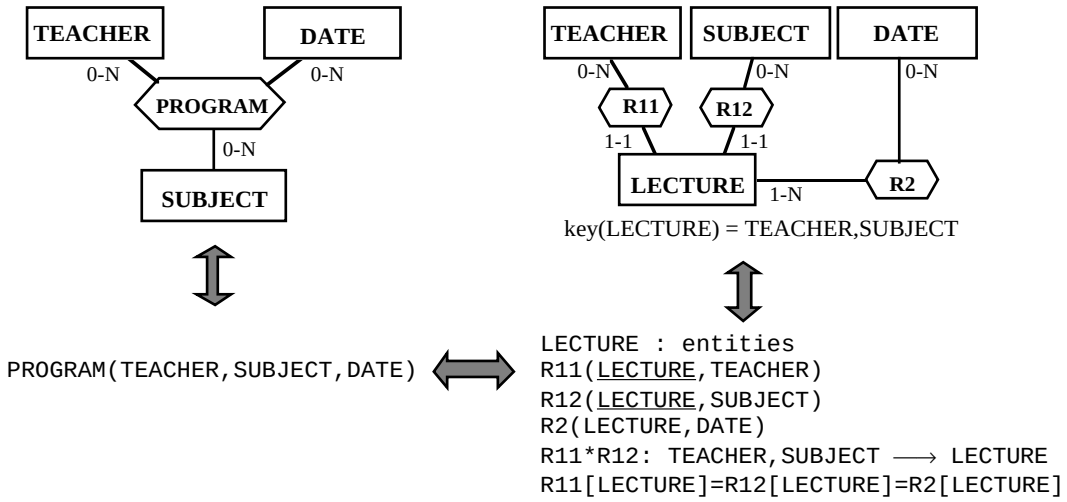
4.2 The extension transformation of entity-based schemas

Applying the extension transformation to entity-based models, and specially E-R models, is now fairly straightforward, since it can be defined on the GER expression of entity-based schemas. A reverse interpretation of the target schemas into the entity-based formalism completes the process. In this context, the new domain `X` is an entity domain. Therefore the extension transformation adds a new entity type, or removes an existing one in an E-R schema without loss of semantics.

---

9 The graphical conventions are as follows : the role name is written close to the corresponding edge, it is omitted when it the same as the name of the entity type in non-recursive relationship types (as in this schema); the cardinality constraint of each role specifies the *min* and *max* numbers of relationships in which an entity may have this role (N stands for *infinity*).

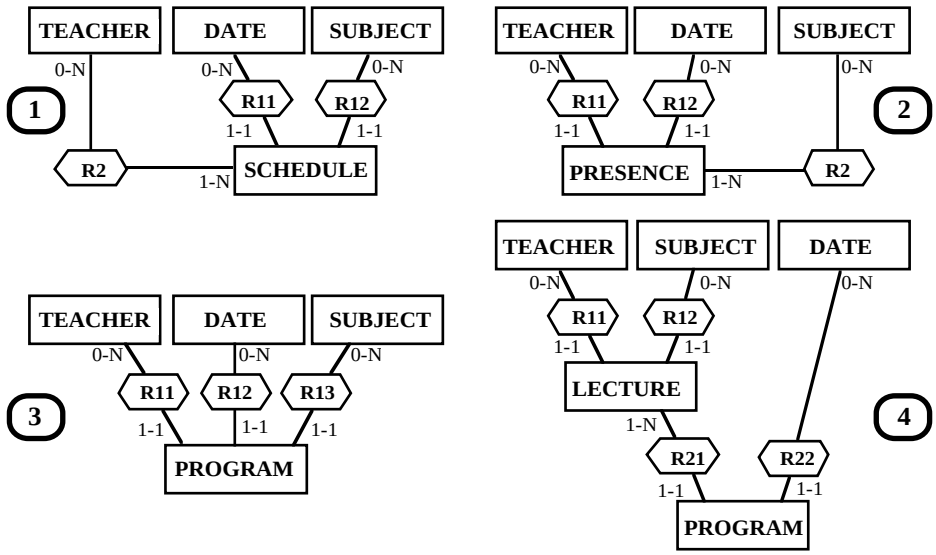
As a first example, let's consider the transformation of the PROGRAM schema developed at the end of section 3.2. Schema 4.4 illustrates this transformation in the context of the E-R model by considering entity types TEACHER, SUBJECT and DATE, together with the relationship type PROGRAM as described in Schema 4.2. The specialization is made according to the substitution `I={TEACHER,SUBJECT}`. Semantically speaking, that choice of I makes the elementary relationship between a given *teacher* and a given *subject* explicit. That concept may correspond to a *lecture*, i.e. the fact that a subject is developed by a teacher, independently of the date. Schema 4.4 exhibits the formal process : (1) expression E-R → GER, (2) extension transformation[10] of the source GER schema into the target GER schema, (3) interpretation GER → E-R. Note that, due to the SR-property of the transformation, schema 4.4 can be read from left to right but also from right to left.



Schema 4.4 *- An extension transformation applied to an E-R schema, via its relational GER expression. Only the elements relevant to the transformation have been included in the GER schemas. Considered from left to right, the extension transformation is based on* `I = {TEACHER,SUBJECT}`*, making the concept of lecture explicit. Note that the FD of the target GER schema has been translated into a E-R key consisting of two entity types linked to* `LECTURE`*.*

Choosing other substitution for `I` would lead to different target schemas, all of them being equivalent to schema 4.2. Schema 4.5 gives the graphical representation of some of them. The GER expressions, the transformation steps, as well as the derived integrity constraints have been dropped for conciseness. This figure illustrates the process of equivalent schema generation.

[10] For simplicity, the instance mappings t_1 and t_2 will be dropped from now on, so that we will mainly concentrate on mapping T by describing the source S and target T(S) schemas only.



Schema 4.5 *- Graphical representation of some target schemas obtained by additional transformations of the* `PROGRAM` *relationship type of schema 4.2. Schema* ***1*** *is obtained for (Schema 4.5 ...)* `I={DATE,SUBJECT};` *schema* ***2*** *for* `I={TEACHER,DATE};` *schema* ***3*** *for* `I={TEACHER,DATE, SUBJECT}`*. Schema* ***4*** *is obtained from target schema 4.4 (right) for* `I={LECTURE,DATE}` *applied to* `R2`*. The new entity types have been given names that evoke their semantics.*

5. ANALYSIS OF THE E-R INTERPRETATIONS OF THE TRANSFORMATION

In this section, we shall reexamine some traditional E-R schema transformations in the framework established so far. We shall show how these transformations can be specified as specializations of the extension transformation by *choosing specific substitution patterns for generic variable* `I` *of predicate* `P`. The interest of this analysis is threefold :

• it gives a sound theoretical basis to many popular schema restructuring techniques;

• it establish a complete definition of these techniques, including all the derived integrity constraints;

• it allows the production of new transformations.

Let's first observe that,

1. the GER relation schema describing *entity type E* has the following general structure,

> **R(E,A,RE)**, where
> - E is the name of an entity domain,
> - A is a set of names of attributes defined on value domains,
> - RE is an optional set of names of attributes defined on entity domains,
> - $A \cup RE \neq \{\}$

A is the expression of the attributes of E, if any, while RE corresponds to the concise notation of some functional relationship types. R can have alternate keys comprising elements of A and/or RE.

2. the GER relation schema describing *relationship type R* has the following general structure,

> **S(RE,A)**, where
> - RE is a set of names of attributes defined on entity domains,
> - A is an optional set of names of attributes defined on value domains,
> - $|RE| \geq 2$

RE is the expression of the roles of R while A corresponds to the attributes of R, if any. S must have at least one key, comprising elements of RE and/or A.

We shall only sketch the formal expression of the transformation by giving the GER expression of the source schema, the structure of the substitution for I, and the main parts of the GER expression of the target schema. In GER expressions, `Ai` is an element of `A`, while `Ei` is an element of `RE`.

TRANSFORMATION OF ENTITY TYPES

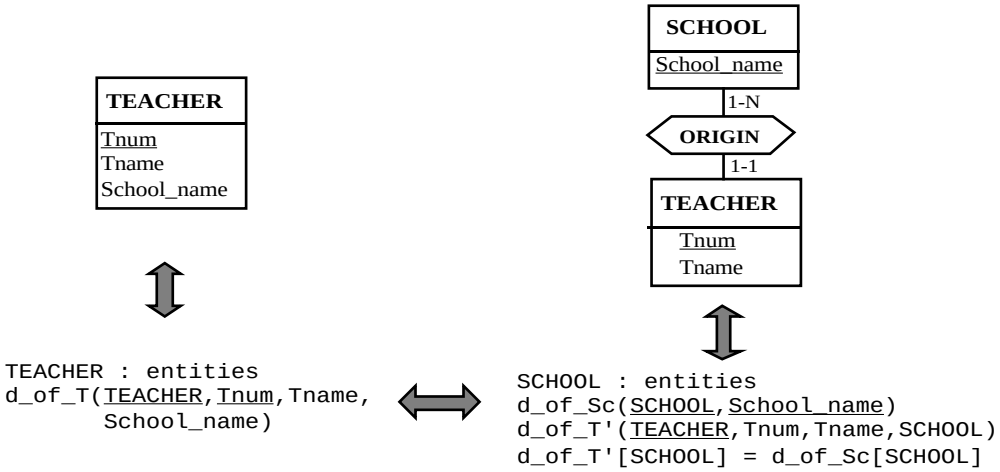
TE1: Transforming a simple attribute into an entity type

Source schema : `R(E,..,Ai-1,Ai,Ai+1,..)`

Substitution : `I = {Ai}`

Target schema : `R'(E,..,Ai-1,X,Ai+1,..); S(X,Ai); S[X]=R'[X]`

Example :



Schema 5.1 *- Attribute* School_name *of entity type* TEACHER *gives birth to entity type* SCHOOL *(left to right). Conversely,* TEACHER *and* SCHOOL *can collapse if* SCHOOL *is linked to* TEACHER *only (right to left).*

TE2: Transforming a multivalued attribute into an entity type

Source schema : R(E,..,Ai-1,Ai[*],Ai+1,..)
(in each tuple of an instance of R, the value of attribute Ai is a set of domain values).

Pre-processing : R is first normalized in 3NF according to Ai, which leads to schema
R'(E,..,Ai-1,Ai+1,, ..); S(E,Ai); S[E]=R'[E]

Schema S can then be transformed according to two strategies as follows :

Substitution 1 : I = {Ai}

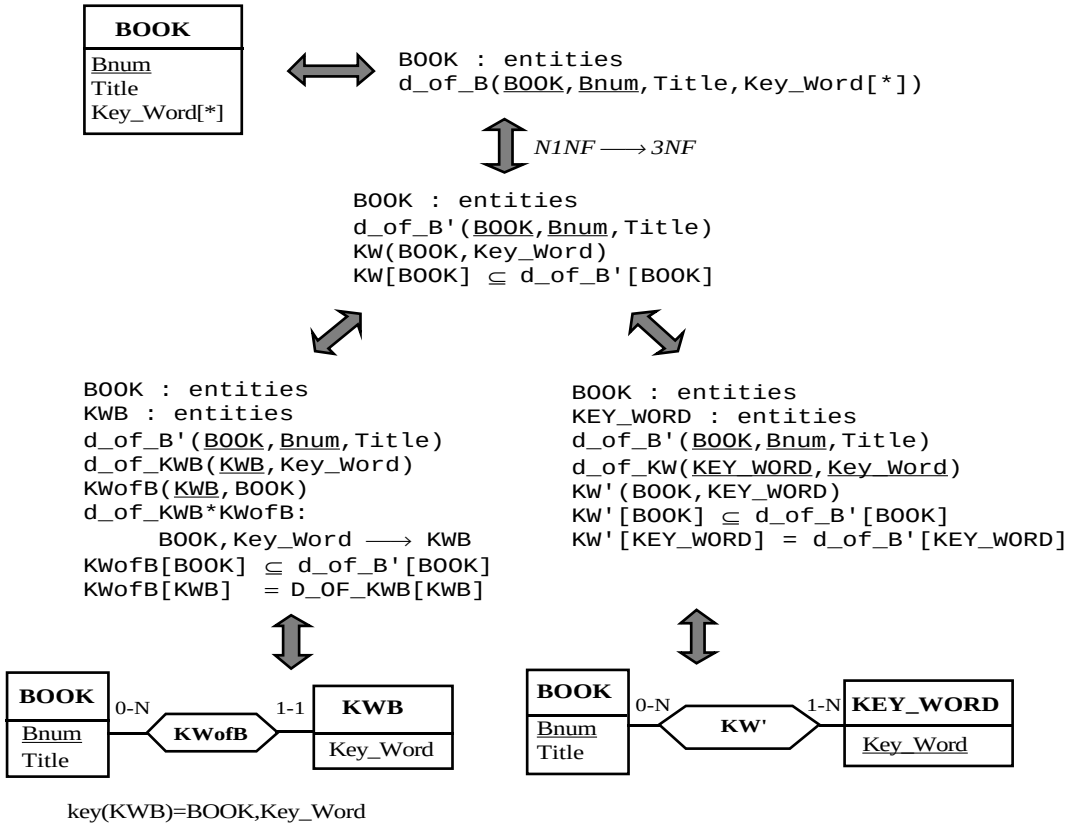
Target schema 1 : R'; S'(E,X); S"(X,Ai); S'[E]=R'[E]; S"[X]=S'[X]

Substitution 2 : I = {E,Ai}

Target schema 2 :
- R'; S'(X,Ai); S"(X,E); S'*S":Ai,E ⟶ X; S'[E]=R'[E]; S"[X]=S'[X] (with post-processing; standard notation for functional rel.types);
- R'; S'(X,Ai,E); S'[E]=R'[E] (basic transformation; concise notation for functional rel.types);

According to the first substitution, an entity X represents a *distinct value* of Ai.
According to the second one, an entity X represents an *occurrence* of Ai in S.

Example :



Schema 5.2 *- Processing a multivalued attribute (denoted by the [*] notation) can be done after normalizing the NF2 GER expression of the entity type attributes. In the example above, multivalued attribute* `Key_Word` *is isolated into all-key relation schema* `KW(BOOK,Key_Word)`*, which will be further transformed. The left transformation of* `KW` *is based on* `I={BOOK,Key_Word}`*, making keyword instances in each book explicit (entity type* `KWB`*, linked to* `BOOK` *through a many-to-one relationship type). The right transformation of* `KW` *is based on* `I={Key_Word}`*, making keywords explicit, independently of their presence in one or several books (entity type* `KEY_WORD`*, linked to* `BOOK` *through a many-to-many relationship type).*

TE3: Transforming a compound attribute into an entity type

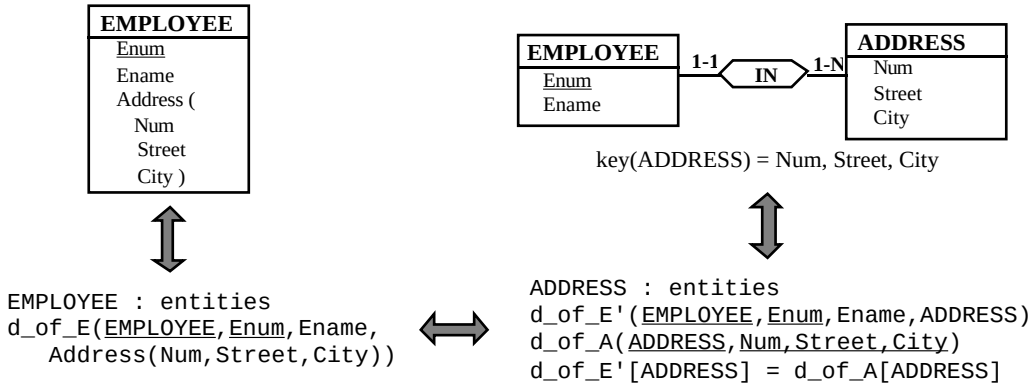
Source schema : `R(E,..,Ai-1,Ai(Ai1,..,Aim),Ai+1,..)`
(in each tuple of an instance of R, the value of attribute `Ai` is a tuple of values of `Ai1,..,Aim`).

Pre-processing : `R` is first normalized into 1NF according to `Ai`, which leads to schema
`R'(E,..,Ai-1,Ai1,..,Aim,Ai+1, ..)`

Substitution : `I = {Ai1,..,Aim}`

Target schema : `R"(E,..,Ai-1,X,Ai+1,..); S(X,Ai1,..,Aim); S[X]=R"[X];`
basic transformation

Example :



Schema 5.3 *- Compound attribute* `Address` *of entity type* `EMPLOYEE` *gives birth to entity type* `ADDRESS` *(left to right). Conversely,* `EMPLOYEE` *and* `ADDRESS` *can merge if* `ADDRESS` *is linked to* `EMPLOYEE` *only (right to left).*

TE4: Splitting an entity type into two entity types (attribute extraction)

Source schema : `R(E,..,Ai-1,Ai,..,Aj,Aj+1,..)`

Substitution : `I = {E,..,Ai-1,Aj+1,..)`

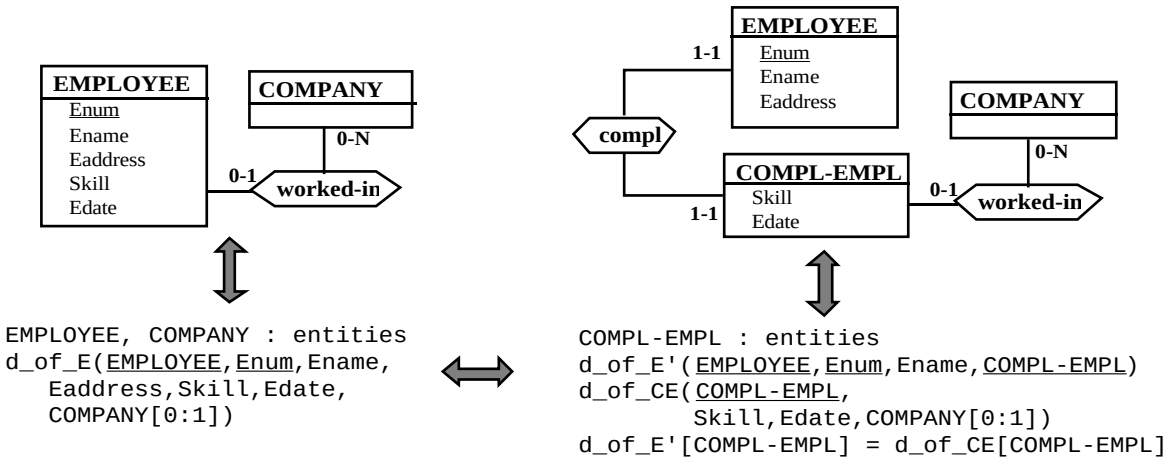
Target schema : `R'(E,..,Ai-1,Aj+1,..,X); S(X,Ai, ..,Aj);R'[X]=S[X];`
basic transformation;

Refinement : By computing the minimal FD, this schema can be simplified as follows :
`R'(E,..,Ai-1,Aj+1,..,X); S(X,Ai, ..,Aj);R'[X]=S[X];`

Interpretation :

The attributes of E have been distributed between E and new entity type X, and a *one-to-one* relationship type has been defined between E and X (concise notation). Note that this transformation is intrinsically different from the previous one, where the relationship type between E and X was many-to-one. Note also that if I includes elements of RE, this transformation carries out the distribution of the roles of E as well.

Example :



Schema 5.4 *- Attributes* `Skill` *and* `Edate` *of entity type* `EMPLOYEE`*, together with its parnership in relationship type* `worked-in` *are extracted to form the new entity type COMPL-EMPL (left to right). Conversely,* `EMPLOYEE` *and* `COMPL-EMPL` *can merge since they are connected by a one-to-one relationship type (right to left).*

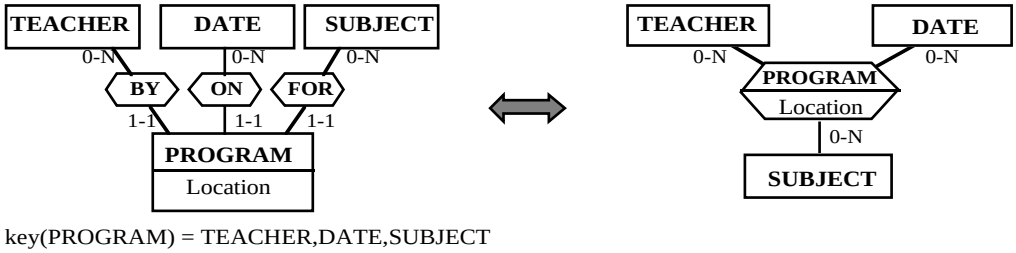
TE5: Transforming an entity type into a relationship type

Source schema : `R'(X,A1,..Am);S1(X,E1);..;Sn(X,En);S1[X]=..=Sn[X]=R'[X];`
`S1*..*Sn:E1,..,Ek ⟶ X;`

Target schema : `R(E1,..En,A1,..Am);`
the key of R is `{E1,..Ek}`

Note : This transformation is the inverse of TR1

Example : The following example, briefly sketched, is borrowed from Schema 5.8, read from right to left.



Schema 5.5 *- Entity type PROGRAM (left) is transformed into rel. type PROGRAM (right).*

TE6: Normalizing an entity type

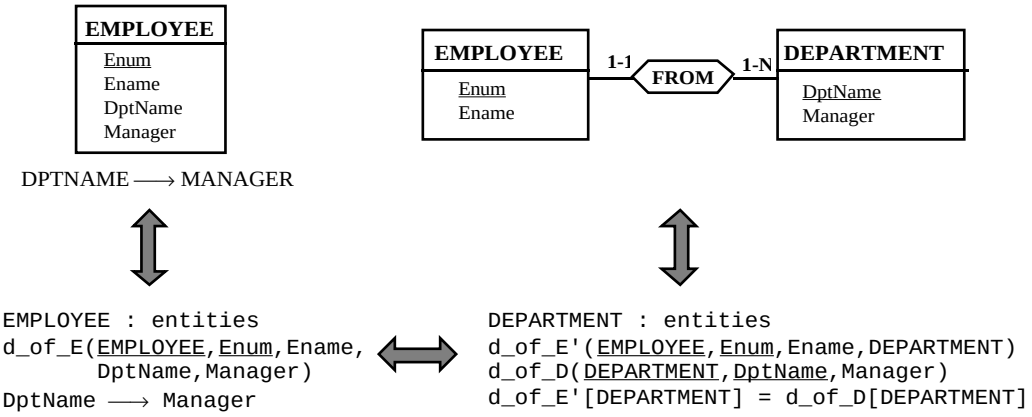
Source schema : `R(E,..,Ai,..,Aj,Ak,..,Al,..); Ai,..,Aj ⟶ Ak,..,Al;`
`Ai,..,Aj` does not include a key of R

Substitution : `I = {Ai,..,Aj,Ak,..,Al}`

Target schema : `R'(E,..,Ai-1,X,Al+1,..); S(X,Ai,..,Aj,Ak,..,Al); R'[X]=S[X]`

Generalization : the principles remains valid when the FD includes elements of RE as well;

Example :



Schema 5.6 *- The set of attributes of EMPLOYEE is not in 3NF due to the presence of a transitive FD. That FD is extracted by an extension transformation based on the attributes participating in the FD: I={DptName,Manager}.*

TE7: Partitioning an entity set through a relationship type

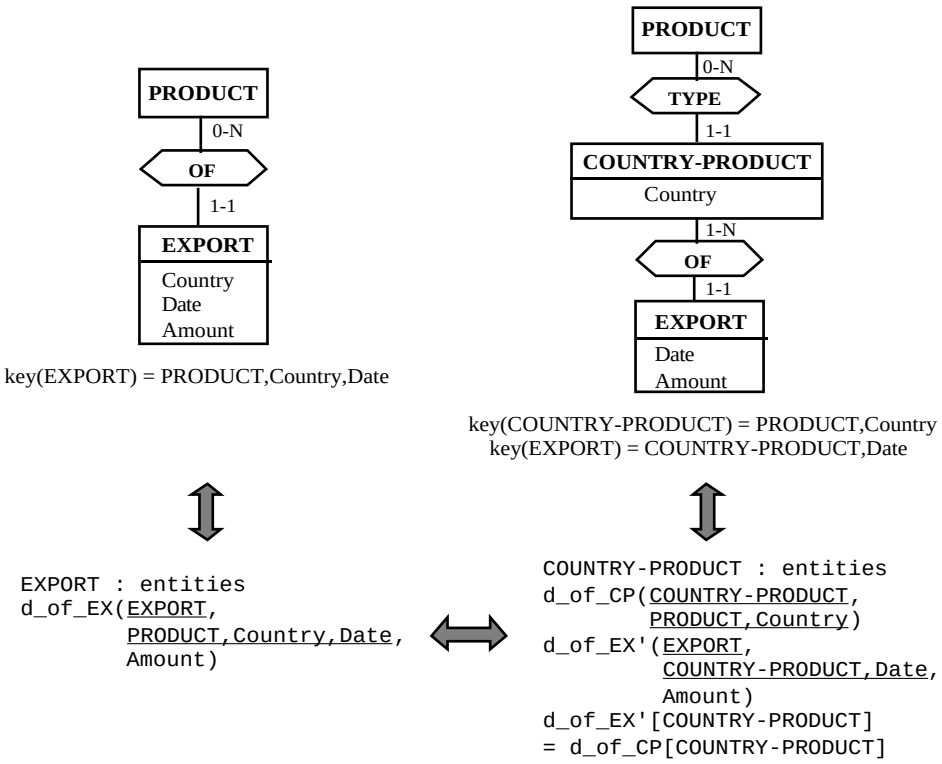
This transformation is rather uncommon and will be considered as new by most designers.

Source schema : `R(E,..,Ai,..,Aj,..,Ek,..);`

Substitution : `I = {Ai,..,Aj,Ek}`

Target schema : `R'(E,..,Ai-1,Aj+1,..,Ek-1,Ek+1,X,..); S(X,Ai,..,Aj,Ek); R'[X]=S[X]`

Example :



Schema 5.7 *- The EXPORT entities of a given PRUDUCT are partitioned according to the destination COUNTRY. A COUNTRY-PRODUCT entity represents a PRODUCT in a given (Schema 5.7 continued) COUNTRY (`I={PRODUCT,Country}`). Note the propagation and translation of the key of `d_of_EX`.*

TRANSFORMATION OF RELATIONSHIP TYPES

TR1: Transforming a relationship type into an entity type

Source schema : `R(E1,..En,A1,..Am)`

`{E1,..En} includes a key of R`[11]

Substitution : `I = {E1,..En}`

Target schema : `R'(X,A1,..Am);S1(X,E1);..;Sn(X,En);S1[X]=..=Sn[X]=R'[X];`

`S1*..*Sn:E1,..,En ⟶ X;`

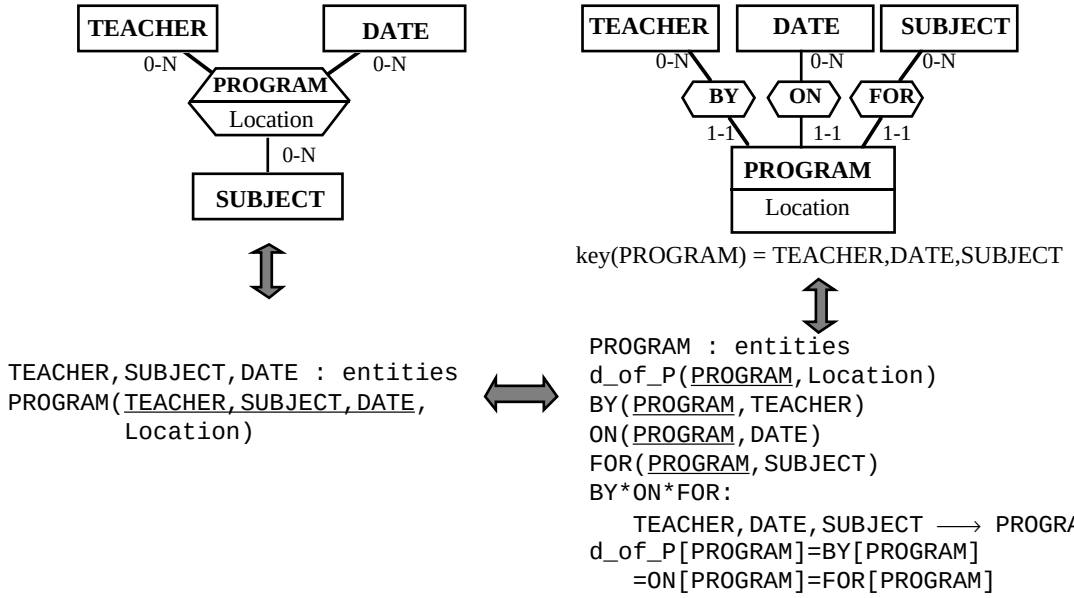
Note : if the key of R is RE', a strict subset of RE, then the determinant of the FD should be replaced by RE';

Generalization : • `I` can include attributes as well; ideally, I will include the key of R (see 3.3);

• `I` can include a strict subset of RE (see TR5);

[11] This constraint is not necessary but it represents the current state of practice.

Example :



Schema 5.8 *- Relationship type* PROGRAM *is transformed into entity type* PROGRAM *(left to right). The inverse transformation (TE5) is sketched in Schema 5.5 (right to left).*

TR2: Eliminating a many-to-many relationship type

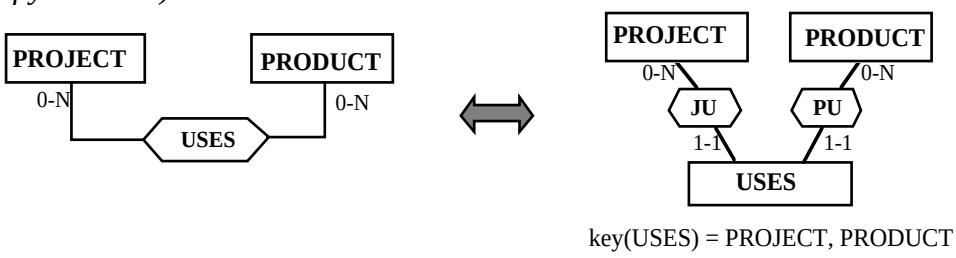
This transformation is a special case of TR1. It is one of the mostly used transformation (creation of a *link entity type*) :

Source schema : `R(E1,E2)`

Substitution : `I = {E1,E2}`

Target schema : `S1(X,E1);S2(X,E2); S1[X]=S2[X]; S1*S2:E1,E2 ⟶ X;`

Example (briefly sketched) :



Schema 5.9 *- The many-to-many relationship type* USES *is transformed into entity type* USES *(left to right).*

TR3: Eliminating a recursive binary relationship type

This transformation is a special case of TR1 (substitution 1) or TR5 (substitution 2) :

Source schema : `R(r1:E,r2:E)`

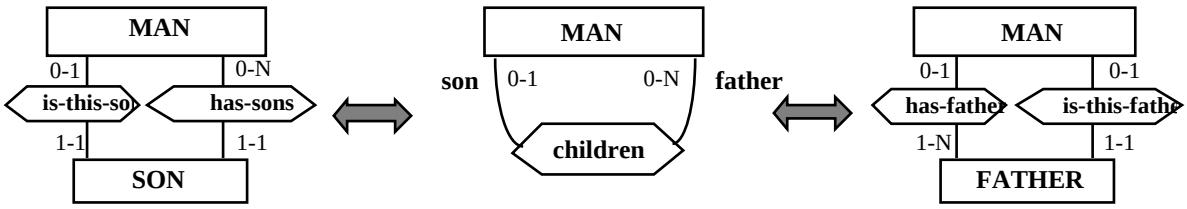
Substitution 1: `I = {r1,r2}` (mainly for N-N rel. types)

Target schema 1: `S1(X,E);S2(X,E); S1[X]=S2[X]; S1*S2:S1.E,S2.E ⟶ X;`

Substitution 2: `I = {r1}` (or `{r2}`) (mainly for non N-N rel. types)

Target schema 2: `R'(X,r2:E);S(X,E); S[X]=R'[X];`

Examples (briefly sketched) :



Schema 5.10 *- The recursive relationship type* `children` *(middle) is transformed (1) into entity type* `SON` *(left) through substitution* `I={son}`*, (2) into entity type* `FATHER` *(right) through substitution* `I={father}`*. From the semantic viewpoint,* `SON` *entities represent* `MAN` *entities who play role* `son` *in* `children`*, while* `FATHER` *entities represent* `MAN` *entities who play role* `father` *in* `children`*.*

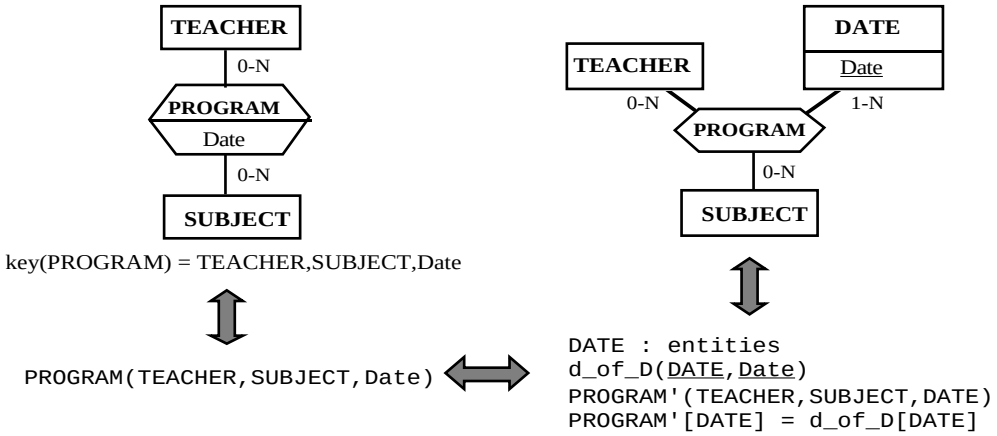
TR4: Transforming relationship type attributes into a role

Source schema : `R(E1,..En,..,Ai,..,Aj,..)`

Substitution : `I = {Ai,..,Aj}`

Target schema : `R'(E1,..En,X,..,Ai-1,Aj+1,..); S(X,Ai,..,Aj); S[X]=R'[X];`

Example :



Schema 5.11 *- Attribute* `Date` *of* `PROGRAM` *is given the status of entity type* `DATE` *(left to right). Conversely, reducing entity type* `DATE` *to a mere attribute reduces the degree of (Schema 5.11 continued)* `PROGRAM` *(right to left). Note that the E-R model used here allows the key of a relationship type to include both roles and attributes, a feature that is easy to accept when considering the GER expression of a relationship type.*

TR5: Aggregating a subset of roles

This transformation is rather uncommon and will be considered as new by most designers.

Source schema : `R(..,Ei,..,Ej,..)`

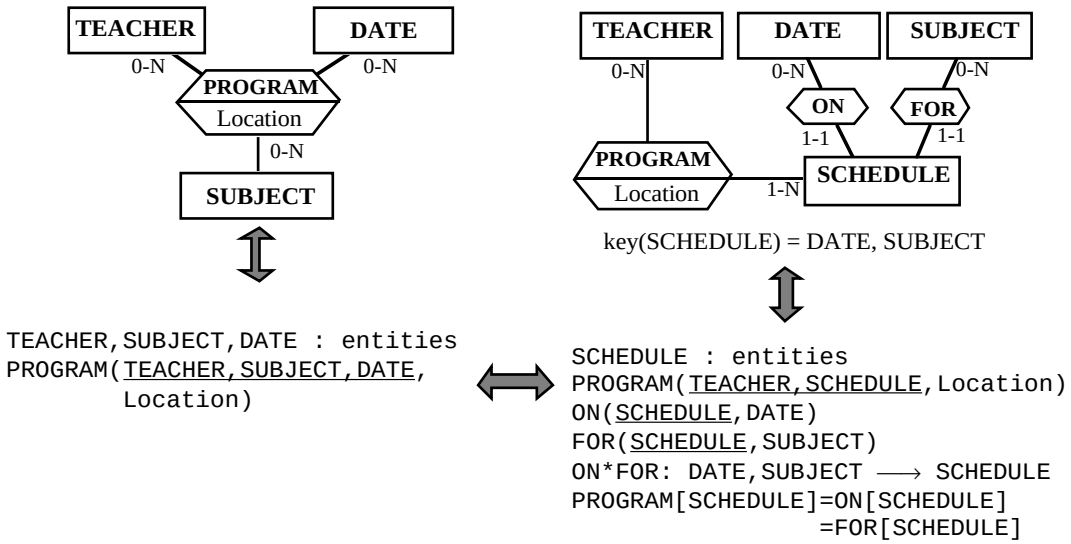
Substitution : `I = {Ei,..,Ej}`

Target schema : `R'(..,Ei-1,X,Ej+1,..)`

`Si(X,Ei);..;Sj(X,Ej); Si[X]=..=Sj[X]=R'[X];`

`Si*..*Sj:Ei,..,Ej ⟶ X;`

Example :



Schema 5.12 *- The roles* `DATE` *and* `SUBJECT` *of relationship type* `PROGRAM` *are extracted and represented by entity type* `SCHEDULE` *(left to right). Conversely, entity type SCHEDULE can be merged with relationship type* `PROGRAM` *(right to left).*

TR6: Aggregating a subset of roles and attributes

This transformation is rather uncommon and will be considered as new by most designers.

Source schema : `R(..,Ei,..,Ej,..,Ak,..,Al,..)`

Substitution : `I = {Ei,..,Ej,Ak,..,Al}`

either I is a (subset of a) key of R, or I doesn't overlap with any key;

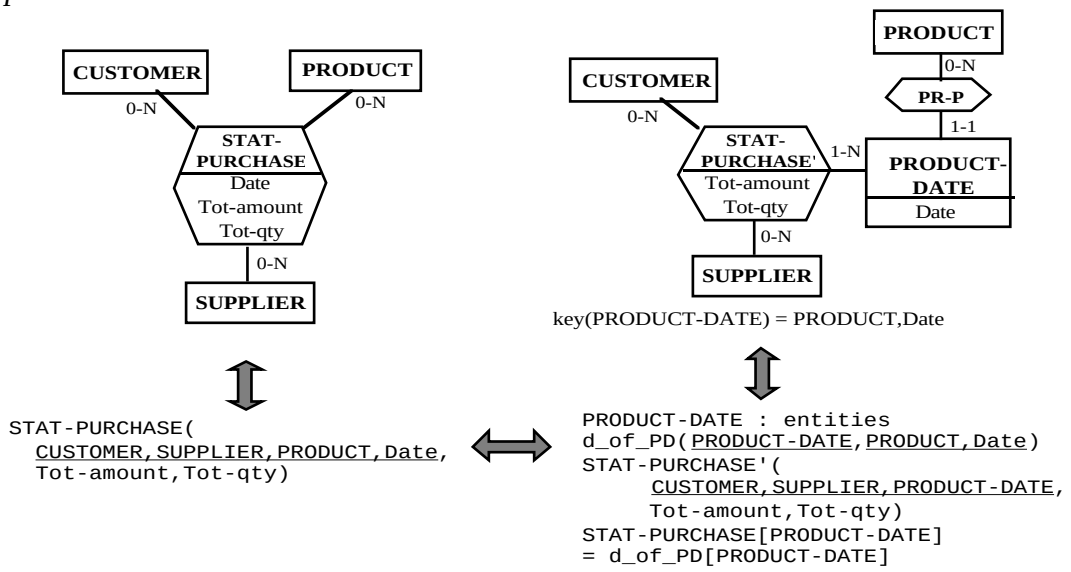
Target schema : `R'(..,Ei-1,..,Ej+1,..,X,..,Ak-1,..,Al+1,..);`

`S(X,Ei,..,Ej,Ak,..,Al);`

`S[X]=R'[X]`

basic transformation; concise notation for functional rel.type;

Example :



Schema 5.13 *- Role* `PRODUCT` *and attribute* `Date` *are aggregated into entity type* `PRODUCT-DATE` *and extracted from relationship type* `STAT-PURCHASE` *(*`I={PRODUCT,Date}`*).*

6. CONCLUSIONS

As far as the author knows, the first original point of this paper is that it proposes some important basis for a general theory of schema transformation. In particular, it has put forward the twofold aspect of the underlying mapping, both at the instance and at the schema levels, it has distinguished the notions of generic and specific transformations, it has identified the specialization process, and it has distinguished two interpretations of the concepts of reversibility and semantic equivalence. In addition, it proposes an adequate notation for specifying generic and specific transformations.

The second original point, as illustrated in section 5, is the proposal of a very general, though still pragmatic, generic entity-generating and entity-removing transformation that can generate and generalize a fairly large number of common practices in schema restructuring. On the basis of a relational analysis, the formal proof of its reversibility, the precise conditions of applicability and the derived integrity constraints have been simply and clearly stated. Not only it gives a theoretical ground to this large set of pragmatic transformations, but it leads to the discovering of new powerful E-R transformations.

It is clear that the extension transformation alone is not sufficient to cover all needs, and to formalize all practices in schema restructuring. Even several entity-generating transformations are of a different nature (let's only mention building a generalization entity type for a set of entity types). Some other transformations can be defined on the basis of project/join operators [D'ATRI,84] [KOBAYASHI,86]. More comprehensive (but with a less in-depth development) transformation toolsets have been proposed in [KOBAYASHI,86] and [ROSENTHAL,88].
It is important to note that the extension transformation is aimed at formalizing, analyzing and generalizing common practices, and at discovering new ones. It is in no way a transformation tool directly intended for practitioners. As an example, the University of Namur has developed a CASE tool for database design based on a transformational approach [CADELLI,90]. In this tool, the extension transformation has been specialized into three pragmatic transformation families that match concrete situations that frequently occur in database design : transforming a relationship type into an entity type, transforming an attribute into an entity type and splitting the attributes of an entity type into two entity types.

This paper lets aside the update propagation between instances of equivalent schemas, an aspect which is important in view derivation and multibase problems. In addition, the problem of constraint propagation is only sketched for FD ([KOBAYASHI,86] gives some hints as to its complexity). Transformations are not mere mind constructs, but are practical goal-oriented tools. Except in some examples in section 5, this paper is not concerned with the rules that should be used to choose the best transformation in specific situations. Finally, important aspects such as the behaviour of formulae and meta-formulae in specializing generic transformations should be developed further.

7. ACKNOWLEDGEMENTS

I want to express my thanks to the reviewers for their constructive comments and suggestions. However, I got them too late to take all of them into account in this version of the paper.

8. REFERENCES

ABITEBOUL,87, Abiteboul, S., Beeri, K., "On the power of languages for the manipulation of complex objects", INRIA technical report, 1987

BATINI,83, Batini, C., Lanzerini, M., Moscarini, M., "View integration", in Methodology and tools for data base design, Ceri (Ed.), North-Holland, 1983

BEERI,86,Beeri, K., Kiefer, "An integrated approach to logical design of relational database schemes", ACM TODS, Vol. 11, N° 2, 1986

BERT,85, Bert, M., N., and al., "The logical design in the DATAID Project : the EASYMAP system", in Computer-Aided Database Design : the DATAID Project, Albano and al. (Ed.), North-Holland, 1985

CADELLI,90, Cadelli, M., Decuyper, B., Hainaut, J.-L., "TRAMIS, a transformation-based workbench for database design", Technical report, Institut d'Informatique, University of Namur, 1990

CASANOVA,84, Casanova, Amaral de Sa, "Mapping Uninterpreted Schemes into Entity-Relationship diagrams: two applications to conceptual schema design", IBM J. Res. & Develop., Vol.28, N°1, Jan. 1984

D'ATRI,84, D'Atri, A., Sacca', D., Equivalence and Mapping of Database Schemes, in Proc. 10th VLDB conf., 1984

FAGIN,77, Fagin, R., "Multivalued dependencies and a new normal form for relational databases", ACM TODS, Vol. 2, N°3, 1977

FAGIN,81, Fagin, R., "Normal Form for Relational Databases Bases on Domains and Keys", ACM TODS, Vol. 6, N°. 3, September 1981

GIRAUDIN,85, Giraudin, J.-P., Delobel, C., Dardailler, P., "Eléments de construction d'un système expert pour la modélisation progressive d'une base de données", in Proc. Journées Bases de Données Avancées, INRIA, Mars 1985

HAINAUT,81, Hainaut, J.-L., "Theoretical and practical tools for data base design", in Proc. Intern. VLDB conf., ACM/IEEE, 1981

HAINAUT,89, Hainaut, J.-L., "A Generic Entity-Relationship Model", in Proc. of the IFIP WG 8.1 Conf. on *Information System Concepts: an in-depth analysis*, North-Holland, 1989.

HAINAUT,90a, Hainaut, J.-L., "The transformation toolkit of TRAMIS : analysis of pragmatic E-R transformations", Research report, University of Namur, 1990

HAINAUT,90b, Hainaut, J.-L., "Entity-Relationship models : formal specification and comparison a tutorial", in Proc. 9th Int. conf. on E-R Approach, 1990

HULL,87, Hull, "A survey of theoretical research on typed complex database objects", in *International Lecture Series in Computer Science*, Academic Press, 1987

JAJODIA,83, Jajodia, S., Ng, P., A., Springsteel, F., N., "The problem of Equivalence for Entity-Relationship Diagrams", in IEEE Trans. on Soft. Eng., SE-9, 5, Sept. 1983

KOBAYASHI,86, Kobayashi, I., "Losslessness and Semantic Correctness of Database Schema Transformation : another look of Schema Equivalence", Inform. Systems, Vol.11, No.1, 1986

KOZACZYNSKY,87, Kozaczynsky, Lilien, "An extended Entity-Relationship(E2R) database specification and its automatic verification and transformation", in Proc. 6th conf. on Entity-Relationship Approach, 1987

LIEN,82, Lien, Y., E., "On the equivalence of database models", JACM, 29, 2, April 1982

LING,89, Ling,T., W., "External schemas of Entity-Relationship based DBMS", in Proc. of the 7th conf. on Entity-Relationship Approach, North-Holland, 1989

MAIER,83, Maier, "The Theory of Relational Databases", Computer Science Press, 1983

MOTRO,87, Motro, "Superviews: Virtual integration of Multiple Databases", IEEE Trans. on Soft. Eng. SE-13, 7, July 1987

NAVATHE,84, Navathe, S., B., Sashidhar, T., Elmasri, R., Relationship Merging in Schema Integration, in Proc. 10th VLDB conf., 1984

REINER,86, Reiner, D., and al, "A Database Designer's Workbench", in Proc. of the 5th conf. on Entity-Relationship Approach, 1986

RISSANEN,77, Rissanen, "Independent components of relations", ACM TODS, Vol. 2, N°4, 1977

ROSENTHAL,88, Rosenthal, Reiner, "Theoretically sound transformations for Practical Database Design", in Proc. of the 6th Int. Conf. on Entity-Relationship Approach, March (Ed.), North-Holland, 1988

ULLMAN,88, Ullman, "Principles of database and knowledge-base systems", Computer Science Press, 1988

APPENDIX. The GENERIC ENTITY/RELATIONSHIP model

The GER model is a **reference model** the aim of which is to describe, specify and compare most current information structure specification formalisms. Such models as E-R (basic and extended), N1NF, Category, binary (NIAM for instance), complex-object, object-oriented (at least for their static components), and DBMS data structures can be defined as specializations of the GER model. The GER model is based on an *extended relational model* the main characteristics of which are as follows.

1. The concepts of the GER model are the *domains*, the *relation schemas* and the *integrity constraints*.
2. A domain is either a *basic domain*, a *subdomain* or a *constructed domain*.
3. The basic domains are predefined named sets of similar elements : `integer`, `char`, `date`, ..., and `entities`. The latter is the name of the *entity domain*, the elements of which can be used to denote real world objects.
4. A *subdomain* is any named subset of a domain.
5. A *constructed domain* is built by a domain constructor. There are four domain constructors, namely the *cartesian*, the *powerset*, the *algebraic* and the *list* constructors.
6. The *cartesian constructor* defines the cartesian product of one or several domains. Each occurrence of a domain in the cartesian product is called an *attribute* of the product; it is given a name.
7. The *powerset constructor* defines the set of all the subsets of a domain. A constraint can be stated about the minimum and maximum size of these subsets.
8. The *algebraic constructor* defines a set of elements as the result of the evaluation of an algebraic expression on domains. Relational algebraic operators such as union, difference, project, select and join, can be used.
9. The *list operator* defines a set of elements comprehensively by a list of their denotations.
10. A *relation schema* is a named cartesian constructed subdomain.
11. The main *integrity constraints* are the *key constraints*, the *dependency constraints* and the *inclusion constraints*.

A GER schema specifies entity types by *entity domains* (i.e. subsets of ***entities***), relationship types by *relationship relation schemas*, and attributes of entity types by a *descriptive relation schema*. Schemas 4.1, 4.2 and 4.3 are some examples of GER schema descriptions, together with their E-R representations. The main mapping rules between E-R and GER that have been used in these schemas can be sketched as follows.

Entity-Relationship	**GER**
Entity type E	User-defined *entity domain* E
Single-valued attributes A_i of entity type E	*Descriptive relation schema* based on E and A_i, with key E, plus the keys of entity type E
Relationship type R with roles r_i taken by entity types E_i and with attributes A_j	*Relationship relation schema* with attributes r_i (with domains E_i), and attributes A_j; the keys are those of R
Functional binary relationship type with roles r_1 (taken by E_1) and r_2 (taken by E_2) such that r_1 has cardinality *i-1*	Either as above, or : attribute r_2 with domain E_2 added to the descriptive relation schema of entity type E_1; adjust keys according to those of E_1

The GER model associates with most entity-based models a relational interpretation that makes many results of the relational theory valid in these models. The *extension transformation* presented in this paper is only one example. An extended description of the GER model and of some of its most common specializations can be found in [HAINAUT,89] and [HAINAUT,90].