# Transformation of Database Populations and Operations from the Conceptual to the Internal Level

P. van Bommel * Gy. Kovács† A. Micsik †

April 18, 1994

**Published as:** P. van Bommel, Gy. Kovács, and A. Micsik. Transformation of database populations and operations from the conceptual to the internal level. *Information Systems*, 1994. In press.

**Abstract**

In this paper we describe a framework for the automatic transformation of databases from the conceptual to the internal level. Our approach fits in with the ANSI/SPARC three level architecture for database management systems.

The framework consists of three components: structures (database schemata), populations (database contents) and operations (database manipulations). On the conceptual level we describe these components for modelling techniques, such as ER, NIAM and the Binary Relationship Model. On the internal level we focus on (nested) relational models.

For each component on the conceptual level a separate transformation is provided to the internal level. The components are integrated by a central notion: the predicator. Further integration is obtained by means of basic population rules.

## 1 Introduction

Database transformations have been extensively studied in literature (see e.g. [2], [3], [16], [20], [21], [22]). It is found that these transformations involve far more problems than just schema translation. These problems are further increased by the fact that very often the source database model is a conceptual model, which is especially suited to express data semantics, while the target data model is a (traditional) internal model, specifying how the database is actually represented in a computer system (cf. [15]).

As conceptual and internal models are used for different purposes, their capabilities are also quite different. We summarize the possibilities of both kinds of models. Conceptual models directly represent the semantics of the data to be handled by the system (e.g. [8], [12], [21], [24]). For this purpose a large variety of constraints is used along with generalization and specialization of object types and objectification of fact types (relationship types). Furthermore, high level information languages may be used for data definition and manipulation (see e.g. [11], [17], [14]).

Internal models represent the storage structure of the database (see e.g. [4], [26], [28] ). As a consequence these models deal with (nested) tables, data dependencies, access path selection, hashing, B-trees and query optimization. Though data redundancy is considered harmful on the conceptual level, it may be a desirable property of internal models for speeding up the retrieval process.

*Dept. of Information Systems, Faculty of Mathematics and Informatics, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, E-mail: pvb@cs.kun.nl

†Computer and Automation Institute, Hungarian Academy of Sciences, P.O.Box 63, H-1502 Budapest, Hungary. E-mail: {kgy,mics}@ilab.sztaki.hu. This work has been partially supported by the Trans-European Mobility Scheme for European Communities (TEMPUS).

Unfortunately a general transformation approach treating all conceptual and internal aspects is not (yet) available. It is stressed here that this is *not* the aim of this paper. However, we do aim at a framework for database transformation from the conceptual to the internal level, in which the essential features are captured. We therefore adopt three notions playing a central role in the field of computer algorithms: data structure, data value and data operation (see e.g. [1], [19]). Usually, the conceptual and internal aspects mentioned earlier are considered in relation to these notions. The framework we describe specifies these three notions on both the conceptual and the internal level, and provides a systematic transformation.

The organization of this paper is as follows. In section 2 the set-up of the framework is described. Section 3 focusses on the notion of data structure. We describe an intermediate language between the data structures of existing conceptual and internal data modelling techniques. In section 4 the attention is focussed on the notion of data value. The conceptual and internal structures are now populated (instantiated) and a transformation between these populations is described. Section 5 and 6 treat the notion of data operation. First we define conceptual and internal operations, without defining the relationship between the two levels. Population rules defined in section 4 will be checked for each operation. Next we describe the transformation of conceptual operations into internal operations.

Our approach is based on the Predicator Model ([8]), a formal definition of data models with an underlying object-role structure. For more details about the Predicator Model, see also [6], [9], [10], [18], [29].

# 2 Framework for database transformation

In this section we describe a general framework for database transformation. Let $\Sigma_1$ be a structure description of a database. Suppose we perform a transformation resulting in a new structure description, say $\Sigma_2$. Although different terms may be used for these descriptions, e.g. information structure, database schema or database structure, they have the common property that they focus on types, rather than instances of those types (i.e. populations conforming to the structure description). As a result, we must also consider the consequences of the transformation mentioned above for these instances. More concretely, if $\mathsf{Pop}(\Sigma_1)$ is a population of structure $\Sigma_1$ and this structure $\Sigma_1$ is transformed into $\Sigma_2$, how is a population $\mathsf{Pop}(\Sigma_2)$ of structure $\Sigma_2$ obtained, such that all data in $\mathsf{Pop}(\Sigma_1)$ is represented? For database operations a similar problem occurs.

Usually, databases are considered on different levels of abstraction. In this paper we use the well-known distinction between the conceptual level and the internal level ([15]). The conceptual level focusses on correct specification of semantics, while the internal level focusses on efficient realisation. On both levels we consider structures, populations and operations. We will describe a transformation from the conceptual level to the internal level. The overall transformation process is shown in the activity graph in figure 1. In this graph a box represents an activity, while a circle represents the input or output of an activity.

In figure 1, we see three transformation activities. The middle transformation of the three involves the database structure. A conceptual structure is transformed into an internal structure.

On the left hand side we see the population transformation. Populations of the conceptual structure are transformed into populations of the internal structure at hand. As a consequence, conceptual populations as well as the conceptual structure and the internal structure are used for this transformation.

On the right hand side we see the operation transformation. Conceptual operations are transformed into internal operations. Again, conceptual and internal structures are needed for this transformation.

Next we discuss the use of population rules. A population is not an isolated notion, since it is associated with a structure. Therefore a population must satisfy certain rules. The aim of these rules is to protect the evolution of populations during system usage against unwanted situations. This evolution is caused by (update) operations. As a consequence, for each operation it should be checked under what circumstances these rules are violated. If violation occurs, the operation

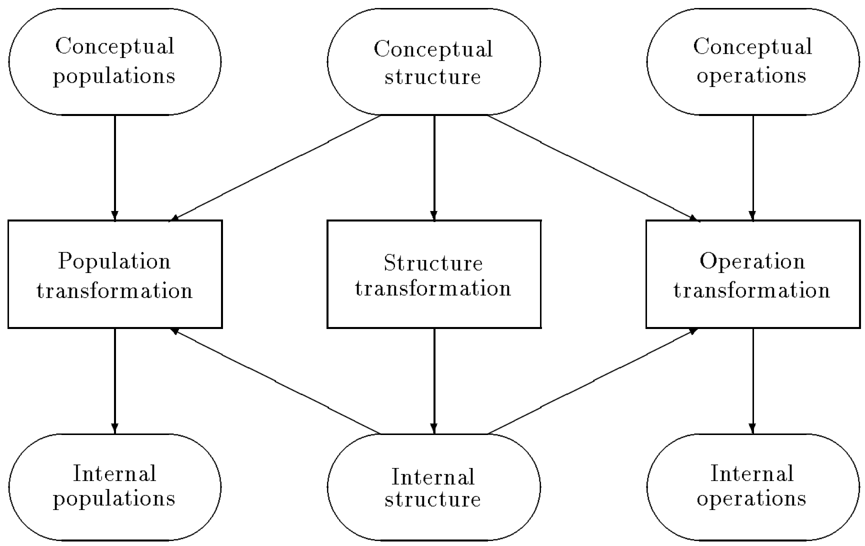


Figure 1: The transformation process

must be rejected or some auxiliary actions have to be undertaken.

In this paper we discuss several basic population rules and consider their violation for each operation we introduce. Usually, more advanced rules can be specified by an information analyst, for example constraints involving uniqueness and mandatory roles, and subset constraints (see [8], [24]). An elaborate treatment of these constraints is outside the scope of this paper. Some aspects will be discussed in section 3.3.

# 3 Conceptual data models and internal representations

In this section we consider the definition of conceptual data models and their internal representations. We first discuss conceptual data models (3.1). Then we define a mechanism for representing such conceptual models on the internal level (3.2).

## 3.1 The conceptual model

We restrict ourselves to conceptual data modelling techniques with an underlying object-role structure (e.g. ER [12], NIAM [21], [24] and the Binary Relationship Model [20], [27]). The information structure in these models consists of the following basic components ([8], [29]):

1. A set $\mathcal{O}$ of object types. For example, in figure 2 we have $A \in \mathcal{O}$. Note that we use $A$ as the designation of a type. As a consequence, $A$ is *not* the set of object instances considered to belong to that type.

2. A set $\mathcal{P}$ of predicators. A predicator specifies the connection between an object type and a role in a fact type. In figure 2 for example we have $p \in \mathcal{P}$. The associated object type is found by the operator $\mathsf{Base} : \mathcal{P} \rightarrow \mathcal{O}$.

3. A set of fact types $\mathcal{F}$, defined as a partition of $\mathcal{P}$. In figure 2 we have $\mathcal{F} = \{f, g, h\}$ where e.g. $f = \{p, q\}$. For description purposes we use $f$ as the designation of fact type $\{p, q\}$. Fact types are object types: $\mathcal{F} \subseteq \mathcal{O}$.

Note that the fact type that corresponds with a predicator can be derived by the operator $\mathsf{Fact} : \mathcal{P} \rightarrow \mathcal{F}$, defined by: $\mathsf{Fact}(p) = f \Leftrightarrow p \in f$. Furthermore both object type and role are specified by a predicator, although for transformational purposes only the object type is necessary. Role names are especially interesting for conceptual data manipulation languages, such as RIDL [14] and LISA-D [17].

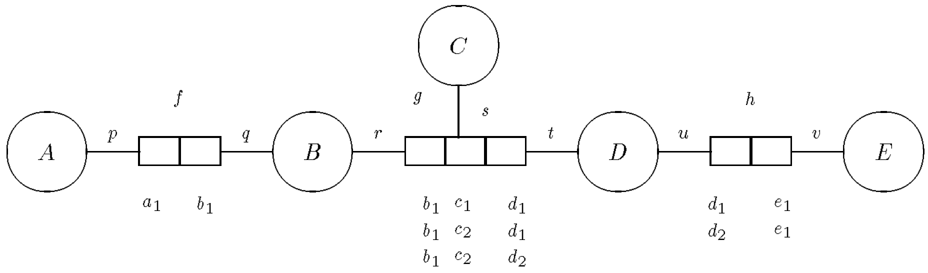


Figure 2: Example information structure with population

Object types can be populated with instances. As an example, in figure 2 we see $\langle d_1, e_1 \rangle$ as an instance (tuple) of fact type $h$. Formal definitions and notational conventions for populating (instantiating) information structures are given in section 4.1.

## 3.2 Internal representations

In this section we discuss a mechanism for representing conceptual information structures internally. Intuitively, such a representation is the result of lifting up certain object types of the information structure and cutting the connections between these object types. Formally, a forest (set of trees) $T = \langle \mathcal{N}, E, \ell \rangle$ is called a tree representation of information structure $\mathcal{I}$, if it is a labelled graph, satisfying the following wellformedness conditions ([9]):

1. The set of nodes $\mathcal{N}$ is a partition of $\mathcal{P}$, s.t. predicators in the same node have the same object type. A node is intended to model a field (or attribute) in the database. In the sequel the following notational convention is used: since $\mathsf{Base}$ is defined on single predicators in section 3.1, the common object type in node $n$ may be denoted as $\mathsf{Base}(n)$.

2. All predicators sharing the same node belong to different fact types.

3. $E$ is a set of edges. The pair $\langle m, n \rangle$ represents an edge from node $m$ to node $n$.

4. Edges are labelled by fact types in the obvious way. The function $\ell : E \rightarrow \mathcal{F}$ assigns $\ell(\langle m, n \rangle) = f$, iff both the source node and the destination node contain a predicator from $f$.

5. Fact types are located around a single parent: $\ell(\langle m_1, n_1 \rangle) = \ell(\langle m_2, n_2 \rangle) \Rightarrow n_1 = n_2$.

These conditions will guarantee the wellformedness of a tree representation with respect to a specific information structure. They do not specify the generation process of tree representations. In [9] a generation algorithm was described. This algorithm yields all possible tree representations for a given information structure. Several guidance parameters were introduced in order to exclude candidates with undesirable properties from the generation process.

**Example 3.1** *In figure 3 we see a possible tree representation of the information structure in figure 2.*

Note that several other tree representations are possible for the same information structure. Note furthermore that the tree representation in figure 3 consists of only one tree. In general, a tree representation consists of a number of trees. In [9] it was shown that the following property can

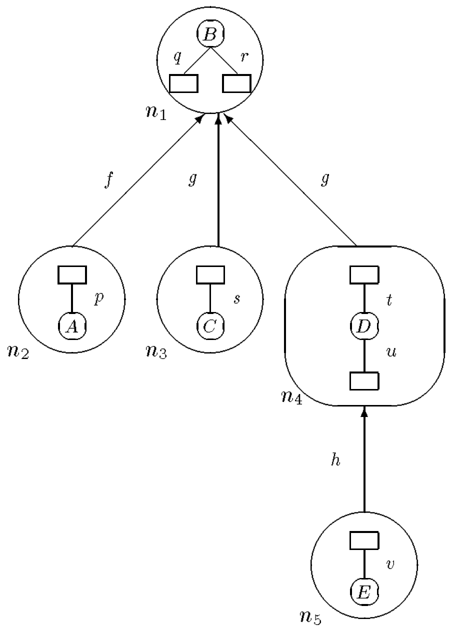


Figure 3: Example tree representation

be derived from the wellformedness conditions discussed above, in combination with the operator $\mathsf{Fact}$:

$$\text{Each edge } e = \langle m, n\rangle \text{ with } \ell(e) = f \text{ has } |m \cap f| = |n \cap f| = 1$$

The corresponding predicator in source node $m$ is unique for node $m$ and is denoted as $\mathsf{Anchor}(m)$. The corresponding predicator in destination node $n$ is unique for fact type $f$ and is denoted as $\mathsf{Hook}(f)$. These notions *anchor* and *hook* will facilitate the transformation of populations and operations in the next sections.

**Example 3.2** *Consider edge* $\langle n_4, n_1\rangle$ *in the tree representation in figure 3. For fact type* $g$ *we have* $\mathsf{Hook}(g) = r$. *For node* $n_4$ *we have* $\mathsf{Anchor}(n_4) = t$.

The layout within a node is defined by the properties of anchors and hooks. A 'top' in a node is an anchor, and thus there can be only one 'top' per node. A 'bottom' in a node is a hook. Obviously there may be several hooks in the same node.

In [6] and [10] it was shown that our concept of tree representations is similar to the concept of nested relations (also called non first normal form or $\mathsf{NF}^2$), as discussed in [13] and [26]. In order to demonstrate this, we give a practical example.

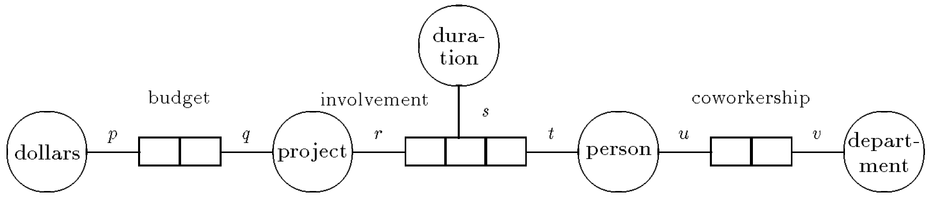


Figure 4: The Project Database

**Example 3.3** *Consider the information structure in figure 4. Suppose that the predicators of this information structure are ordered as prescribed by the tree representation from figure 3. Then, the* $\mathsf{NF}^2$*table structure corresponding to that tree representation is shown in figure 5.*

<table>
<tr><td rowspan="3">project</td><td>budget</td><td colspan="3">involvement</td></tr>
<tr><td rowspan="2">dollars</td><td rowspan="2">duration</td><td rowspan="2">person</td><td>coworkership</td></tr>
<tr><td>department</td></tr>
</table>

Figure 5: A nested table for the Project Database

The (normalized) relational model also fits in our formalism. A normalized relation is simply obtained by flattening a nested relation.

**Example 3.4** *The result of flattening the nested table from figure 5 is shown in figure 6.*

| *project* | *dollars* | *duration* | *person* | *department* |
|---|---|---|---|---|

Figure 6: A relational table for the Project Database

As was mentioned before, different tree representations are possible for one and the same information structure. In this way different nested tables for that information structure can be easily obtained. As an example, the Project Database shown in figure 4 can also be implemented using the tables in figure 7 or figure 8.

<table>
<tr><td rowspan="3">person</td><td>coworkership</td><td colspan="3">involvement</td></tr>
<tr><td rowspan="2">department</td><td rowspan="2">duration</td><td rowspan="2">project</td><td>budget</td></tr>
<tr><td>dollars</td></tr>
</table>

Figure 7: Alternative nested table

### 3.3 Constraints

In this section we discuss two kinds of constraints on the possible populations of information structures: uniqueness constraints and total role constraints. These constraints are of vital importance for the conceptual information structure, since they are used for identification purposes ([8],[24]). Furthermore, these constraints are important for internal representations, because several basic properties of these representations can be derived from them (see e.g. [9]).

First we consider total role constraints. A total role constraint for predicator $p$ expresses that each instance of $\mathsf{Base}(p)$ must occur in at least one instance of $\mathsf{Fact}(p)$. For an internal representation containing predicator $p$ this has the following consequences:

1. If in this representation predicator $p$ is the anchor of a non-leaf node, then predicator $p$ must have a total role constraint. In case this predicator is not total, the internal representation is illegal because it may result in loss of information. Consider for example the situation where predicator $t$ in figure 2 is not total and the population of fact type $h$ also includes instance $\langle d_3, e_3 \rangle$. Then this instance can not be represented in the structure of figure 3.

2. If in this representation predicator $p$ is the hook of a fact type and $p$ does not have a total role constraint, then the subtable corresponding to $\mathsf{Fact}(p)$ is *optional*.

Next we consider uniqueness constraints. The meaning of a uniqueness constraint is similar to the notion of a key (see e.g. [8]). As a consequence, presence or absence of a uniqueness constraint will not cause an internal representation to be illegal. However, a special situation occurs if the hook of a fact type is unique. Then the subtable corresponding to that fact type is flat rather than nested. More details and several examples are found in [9].

<table>
<tr><td rowspan="2">project</td><td>budget</td></tr>
<tr><td>dollars</td></tr>
</table>

<table>
<tr><td rowspan="3">project</td><td colspan="3">involvement</td></tr>
<tr><td rowspan="2">duration</td><td rowspan="2">person</td><td>coworkership</td></tr>
<tr><td>department</td></tr>
</table>

Figure 8: Representation consisting of two nested tables

# 4 Populations

In this section we discuss populations (instantiations) of information structures, tree representations, and their relationship. We first discuss the population of information structures (4.1). Then we introduce populations for tree representations (4.2). Next, we show how a conceptual population (of an information structure) is translated into an internal population (of a tree representation) (4.3).

## 4.1 Information structure population

A population of an information structure $\mathcal{I}$ assigns to each object type in $\mathcal{O}$ a set of values of some universal domain ([23]), conforming to the structure as prescribed by $\mathcal{P}$ and $\mathcal{F}$. The population of an atomic object type is just a set of values. The population of a composed object type (fact type) is a set of tuples. A tuple $t$ of a fact type $f$ is a mapping of all its predicators to values of the appropriate type ([8]):

$$\forall_{f \in \mathcal{F}} \forall_{t \in \mathsf{Pop}(f)} \forall_{p \in f} \left[ t(p) \in \mathsf{Pop}(\mathsf{Base}(p)) \right]$$

This is referred to as the *Conformity Rule*.

**Example 4.1** *Consider the information structure from figure 2. We see that the population of object type $A$ contains one value $a_1$. Furthermore, the population of fact type $f$ contains one tuple $t$, defined as follows:*

$$\begin{aligned} t(p) &= a_1 \\ t(q) &= b_1 \end{aligned}$$

*As a consequence, an alternative denotation for tuple $t$ is $\{(p, a_1), (q, b_1)\}$. When no confusion is likely to occur, $t$ may simply be written as $\langle a_1, b_1 \rangle$ or $\langle b_1, a_1 \rangle$.*

## 4.2 Tree population

In this section we introduce how tree representations are populated. Our way of populating tree representations is analogous with [13], [25] and [26], where the population of a nested relation consists of nested tuples. We describe such tuples in terms of nodes in the forest.

A population of a tree representation $T$ assigns to each node $m \in \mathcal{N}$ a set of values of the population of $\mathsf{Base}(m)$. Let $m \in \mathcal{N}$ be a node. Then, a population $\mathsf{Pop}(m)$ is a set of tuples, where each tuple assigns a unique value to $m$:

$$\forall_{t_1, t_2 \in \mathsf{Pop}(m)} \left[ t_1(m) = t_2(m) \Rightarrow t_1 = t_2 \right]$$

This is referred to as the *Partitioning Rule* for populations (cf. [25]). The value $t(m)$ is called the *root value* of tuple $t$. Note that $\mathsf{Pop}$ is used with two different kinds of arguments. For a conceptual population (of an information structure) the argument is an object type, while for an internal population (of a tree representation) the argument is a node.

Furthermore, a tuple $t \in \mathsf{Pop}(m)$ assigns to each child $n$ of node $m$ a subset of $\mathsf{Pop}(n)$. This assignment takes the structure of fact types into account. Let $f$ be fact type with $\mathsf{Hook}(f) \in m$. The extension of $f$, denoted as $\mathsf{Ext}(f)$, contains all children of node $m$ dealing with fact type $f$.

Let $\mathsf{Ext}(f) = \{n_1, ..., n_l\}$. Then, each $\langle t_1, ..., t_l \rangle \in t(\mathsf{Ext}(f))$ should fit in the population of the corresponding children:

$$\forall_{1 \le i \le l} [t_i \in \mathsf{Pop}(n_i)]$$

This is referred to as the *Fitting Rule*. In the sequel we will use $t(f)$ as a shorthand for $t(\mathsf{Ext}(f))$.

The *Partitioning Rule* and the *Fitting Rule* establish the basis for the mapping of populations from the conceptual level to the internal level.

## 4.3 Transformation of populations

In this section we consider the translation of a conceptual population (of an information structure) into an internal population (of a tree representation). Let $\mathcal{I}$ be an information structure and let $T$ be a tree representation of $\mathcal{I}$. From a population of information structure $\mathcal{I}$ we will generate a population of tree representation $T$.

Let $m$ be a node in $T$. Then, for each instance of the corresponding base a (unique) tuple will result in $\mathsf{Pop}(m)$:

$$\mathsf{Pop}(\mathsf{Base}(m)) = \{\, t(m) \mid t \in \mathsf{Pop}(m) \,\}$$

Next we consider the children of node $m$. Let $f$ be a fact type with $\mathsf{Hook}(f) \in m$. Furthermore let $\mathsf{Ext}(f) = \{n_1, ..., n_l\}$ be the children of node $m$ dealing with fact type $f$. We describe how the internal representation of $\mathsf{Pop}(f)$ is obtained. For each tuple $t \in \mathsf{Pop}(m)$ the assignment $t(n_1, ..., n_l)$ is defined by:

$\langle t_1, ..., t_l \rangle \in t(n_1, ..., n_l)$ iff:

$\langle t(m), t_1(n_1), ..., t_l(n_l) \rangle \in \mathsf{Pop}(f)$

This concludes the transformation of populations from the conceptual to the internal level. Obviously, the resulting population of the tree representation satisfies the *Partitioning Rule* and the *Fitting Rule*. In [10] this was used to estimate the expected storage requirements for a given tree representation.

**Example 4.2** *Let $\mathcal{I}$ be the information structure from figure 2 and let $T$ be the tree representation from figure 3. We transform the population of $\mathcal{I}$ into a population of $T$. Following our definitions, this population will consist of one nested tuple with root value $b_1$. The result of the transformation process is shown in figure 9.*

*Besides root value $b_1$ the nested tuple in figure 9 contains a subtable for the values related to $b_1$ via fact type $f$, and a subtable for the values related to $b_1$ via fact type $g$. The subtable corresponding to fact type $f$ contains a single tuple having only a root value. The subtable corresponding to fact type $g$ consists of three tuples containing a subtable for fact type $h$.*

<table>
<tr><td rowspan="3">$n_1$</td><td>$f$</td><td colspan="3">$g$</td></tr>
<tr><td rowspan="2">$n_2$</td><td rowspan="2">$n_3$</td><td rowspan="2">$n_4$</td><td>$h$</td></tr>
<tr><td>$n_5$</td></tr>
</table>

<table>
<tr><td rowspan="3">$b_1$</td><td rowspan="3">$a_1$</td><td>$c_1$</td><td>$d_1$</td><td>$e_1$</td></tr>
<tr><td>$c_2$</td><td>$d_1$</td><td>$e_1$</td></tr>
<tr><td>$c_2$</td><td>$d_2$</td><td>$e_1$</td></tr>
</table>

Figure 9: Population in nested table

Another (more tree-like) view of the nested tuple in figure 9 is presented in figure 10. It is obtained by lifting up root value $b_1$, incorporating the instances of the associated fact types (fields in the nested table). The complete formal definition of this population is given in example 4.3.

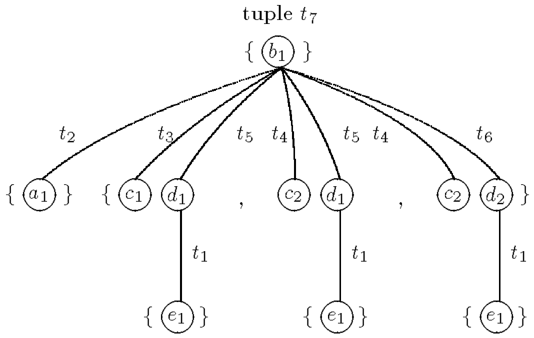


Figure 10: Tree-like view of population

**Example 4.3** *Consider the population shown in figure 10. We formally define this population in terms of nodes of the tree representation in figure 3. The population of the leaves is straightforward:*

$$\begin{array}{lcllrcl} \mathsf{Pop}(n_5) & = & \{t_1\} & \textit{with:} & t_1(n_5) & = & e_1 \\ \mathsf{Pop}(n_2) & = & \{t_2\} & \textit{with:} & t_2(n_2) & = & a_1 \\ \mathsf{Pop}(n_3) & = & \{t_3, t_4\} & \textit{with:} & t_3(n_3) & = & c_1 \\ & & & & t_4(n_3) & = & c_2 \end{array}$$

*For node $n_4$ we have:*

$$\begin{array}{lcllrcl} \mathsf{Pop}(n_4) & = & \{t_5, t_6\} & \textit{with:} & t_5(n_4) & = & d_1 \\ & & & & t_5(n_5) & = & \{t_1\} \\ & & & & t_6(n_4) & = & d_2 \\ & & & & t_6(n_5) & = & \{t_1\} \end{array}$$

*The population of the root $n_1$ consists of tuple $t_7$, defined by:*

$$\begin{array}{lcl} t_7(n_1) & = & b_1 \\ t_7(n_2) & = & \{t_2\} \\ t_7(n_3, n_4) & = & \{\langle t_3, t_5\rangle, \langle t_4, t_5\rangle, \langle t_4, t_6\rangle\} \end{array}$$

# 5 Operations

In this section we will introduce operations for conceptual information structures and internal tree representations. Without losing the generality of our framework for database transformation, we will focus on update operations. We introduce operations for insertion, deletion and modification of populations. For each operation a general definition is given, which is applicable to both the conceptual and the internal level.

Our focus on update operations is motivated as follows. As was mentioned in section 2, the evolution of populations during system usage is caused by update operations. This has several consequences. For example, checking whether population rules are violated should be done when update operations are performed. Furthermore, the set of all possible populations (also called the state space) for a given information structure is induced from the empty population by applying update operations. This also allows for the use of structural induction over the state space. However, in order to shortly address the problem of query transformation from the conceptual to the internal level, a global strategy is discussed in section 6.4.

## 5.1 Insertion

Let $\Gamma$ be a set of tuples (a population) and let $t \notin \Gamma$ be a tuple. Then, the general definition of the insert operation is as follows:

$$\mathsf{Insert}(\Gamma, t) = \Gamma \cup \{t\}$$

First, we consider the insert operation on the conceptual level. Let $f$ be a fact type of our information structure and let $\Gamma = \mathsf{Pop}(f)$. Furthermore, let $t$ be a tuple with domain $\mathsf{Dom}(t)$. For the insertion of $t$ into $\Gamma$ we require $t$ to be well defined: $\mathsf{Dom}(t) = f$.

**Example 5.1** *Consider the information structure and the population from figure 2. The population of fact type $g = \{r, s, t\}$ contains three tuples: $\mathsf{Pop}(g) = \{g_1, g_2, g_3\}$. Suppose we want to insert $\langle b_2, c_1, d_2 \rangle$ into $\mathsf{Pop}(g)$. Then we have to perform $\mathsf{Insert}(\mathsf{Pop}(g), g_4)$, where tuple $g_4$ is defined as follows:*

$$\begin{aligned} g_4(r) &= b_2 \\ g_4(s) &= c_1 \\ g_4(t) &= d_2 \end{aligned}$$

*The result of this operation is $\mathsf{Pop}(g) = \{g_1, g_2, g_3, g_4\}$.*

**Example 5.2** *We give a more practical example in terms of the Project Database shown in figure 4. The insertion from example 5.1 may correspond to the recording of a new involvement:* Insert the fact that person 'Smith' is involved in project 'database design' for the duration of '2 days per week'.

Note that the insertion of a fact instance may violate the *Conformity Rule.* This violation will not occur if tuple $t$ to be inserted uses values of the appropriate type:

$$\forall_{p \in \mathsf{Dom}(t)} [t(p) \in \mathsf{Pop}(\mathsf{Base}(p))]$$

Next, we discuss the insert operation on the internal level. It is important to note that we do not yet consider any relationship with the insert operation on the conceptual level. This relationship will be discussed in section 6.

Let $n$ be a node in a given tree representation and let $\Gamma = \mathsf{Pop}(n)$. For the insertion of tuple $t$ into $\Gamma$ we require this tuple to be well defined:

$$\mathsf{Dom}(t) = \{n\} \cup \{\, \mathsf{Ext}(f) \mid \mathsf{Hook}(f) \in n \,\}$$

where $\mathsf{Ext}(f)$ is the set of nodes, containing all children of node $n$, dealing with fact type $f$.

**Example 5.3** *Consider the tree representation in figure 3 and its population in figure 9. Suppose we want to insert a tuple, say $t$, defined as follows:*

$$\begin{aligned} t(n_1) &= b_2 \\ t(n_2) &= \{t'\} \textit{ where } t'(n_2) = a_1 \\ t(n_3, n_4) &= \emptyset \end{aligned}$$

*The insertion is performed by $\mathsf{Insert}(\mathsf{Pop}(n_1), t)$, resulting in the population shown in figure 11.*

**Example 5.4** *A more practical example is given in terms of the nested table for the Project Database in figure 5. The insertion from example 5.3 may correspond to the recording of a new budget:* Insert the fact that project 'database design' has a budget of 7000 dollars.

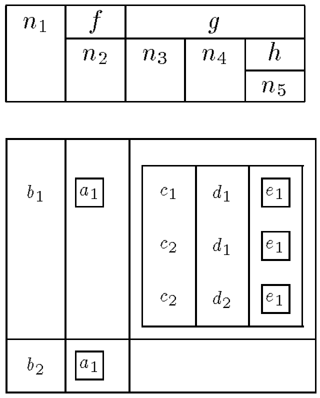


Figure 11: One row has been inserted

Note that the insertion of a tuple $t$ into the population of node $n$ may violate the *Partitioning Rule*. This will not occur if the following condition is satisfied:

$$\forall_{t' \in \mathsf{Pop}(n)} [t'(n) \neq t(n)]$$

The *Fitting Rule* may also be violated by the insertion of tuple $t$ into population $\mathsf{Pop}(n)$. For $\mathsf{Ext}(f) = \{n_1, ..., n_k\} \in \mathsf{Dom}(t)$ this violation will not occur if tuple $t$ uses correct instances of the corresponding children nodes:

$$t(n_1, ..., n_k) \subseteq \prod_{i=1}^{k} \mathsf{Pop}(n_i)$$

## 5.2 Deletion

Let $\Gamma$ be a set of tuples (a population) and let $t$ be a tuple in $\Gamma$. Then, the general definition of the delete operation is as follows:

$$\mathsf{Delete}(\Gamma, t) = \Gamma - \{t\}$$

On the conceptual level this operation is simple. Violation of the *Conformity Rule* will not happen.

**Example 5.5** *Suppose we want to delete the inserted tuple from example 5.1. Then we have to perform* $\mathsf{Delete}(\mathsf{Pop}(g), g_4)$.

Next, we consider the $\mathsf{Delete}$ operation on the internal level, without considering the transformation process from the conceptual to the internal level. This mapping will be discussed in section 6.

**Example 5.6** *Suppose we want to delete the inserted tuple from example 5.3. This deletion is performed by* $\mathsf{Delete}(\mathsf{Pop}(n_1), t)$.

Let $n$ be a node with $\Gamma = \mathsf{Pop}(n)$ and let $t$ be a tuple to be deleted. Obviously, the operation $\mathsf{Delete}(\Gamma, t)$ will not violate the *Partitioning Rule*. However, the *Fitting Rule* may be violated if node $n$ has a father, say $\mathsf{Father}(n)$. This violation does not occur if the father node does not occupy the tuple to be deleted:

$$\forall_{t' \in \mathsf{Pop}(\mathsf{Father}(n))} [t \notin t'(n)]$$

## 5.3 Modification

In this section, we consider the modification of populations. We first introduce the notion of a *partial* tuple. A partial tuple (also called modification function) of a tuple $t \in \Gamma$ is a function m with domain $\mathsf{Dom}(\mathsf{m}) \subseteq \mathsf{Dom}(t)$. For partial tuple m of tuple $t$ the *complementary domain* is defined by:

$$\mathsf{ComDom}(\mathsf{m}) = \mathsf{Dom}(t) - \mathsf{Dom}(\mathsf{m})$$

Partial tuples are used for the definition of *modified* tuples. Such a tuple is defined by: $M(t, \mathsf{m}) = \mathsf{m} \cup t[\mathsf{ComDom}(\mathsf{m})]$. If m is a partial tuple of $t$, then the modify operation is defined as follows:

$$\mathsf{Modify}(\Gamma, t, \mathsf{m}) = \Gamma - \{t\} \cup \{M(t, \mathsf{m})\}$$

First, we consider the modify operation on the conceptual level.

**Example 5.7** *Consider the information structure and the population from figure 2. Suppose we want to modify tuple* $g_1 = \langle b_1, c_1, d_1 \rangle$ *into* $\langle b_1, c_3, d_1 \rangle$. *Then we need a modification function* m *defined by* $\mathsf{Dom}(\mathsf{m}) = \{s\}$ *and* $\mathsf{m}(r) = c_3$. *The actual modification is performed by* $\mathsf{Modify}(\mathsf{Pop}(g), g_1, \mathsf{m})$.

**Example 5.8** *A more practical example is given in terms of the Project Database in figure 4. The modification from example 5.7 may correspond to the following change:* Person 'Blake' being involved in project 'system maintenance' for '2 days per week' will from now on work '4 days per week'.

The modify operation may violate the *Conformity Rule*. Let m be a modification function of tuple $t$. Then, violation is avoided if m uses values of the appropriate type:

$$\forall_{p \in \mathsf{Dom}(\mathsf{m})} [\mathsf{m}(p) \in \mathsf{Pop}(\mathsf{Base}(p))]$$

Next, we consider the modify operation on the internal level. Let $t \in \mathsf{Pop}(n)$ be a tuple in the population of node $n$ and let m be a partial tuple (modification function) of $t$. The modification of $t$ according to m may violate the *Partitioning Rule* if $n \in \mathsf{Dom}(\mathsf{m})$. This will not happen if m has the following property:

$$\forall_{t' \in \mathsf{Pop}(n)} [t'(n) \neq \mathsf{m}(n)]$$

The *Fitting Rule* may also be violated. For $\mathsf{Ext}(f) = \{n_1, ..., n_k\} \in \mathsf{Dom}(\mathsf{m})$ this rule will not be violated if m uses correct instances of the corresponding children nodes:

$$\mathsf{m}(n_1, ..., n_k) \subseteq \prod_{i=1}^{k} \mathsf{Pop}(n_i)$$

In the above definition of the modify operation, we did not make any choice with respect to the implementation of this operation. A straightforward implementation would be as follows:

$$\mathsf{Modify}(\Gamma, t, \mathsf{m}) = \mathsf{Insert}(\mathsf{Delete}(\Gamma, t), M(t, \mathsf{m}))$$

Note that this option uses no intermediate results. As a consequence, this implementation strategy will in general be inefficient. Another option is:

$$\mathsf{Modify}(\Gamma, t, \mathsf{m}) = \mathsf{Delete}(\mathsf{Insert}(\Gamma, M(t, \mathsf{m})), t)$$

This option is quite dangerous. For example, the situation $t = M(t, \mathsf{m})$ will lead to the unwanted deletion of tuple $t$.

# 6 Transformation of operations

In this section we describe the transformation process from conceptual operations into internal operations. For each operation on the conceptual level, we give the necessary activities in terms of internal operations. The operations we use were introduced in section 5.

## 6.1 Transformation of insertion

In this section we consider the transformation of insertions from the conceptual to the internal level. Let $f$ be a fact type. Suppose we want to insert a tuple $t$ into the population of fact type $f$. This insertion is performed by $\mathsf{Insert}(\mathsf{Pop}(f), t)$. Let $T$ be a tree representation in which $f$ is represented internally. We describe how the population of $T$ must be changed. This will happen in two steps:

**proc** TransformInsertion ($f$ : Fact Type; $t$ : Tuple);
  InsertAtomicTuples ($f, t$);
  Connect ($f, t$)
**end**

First the values in $t$ that are new must be inserted. Then they have to be interconnected, such that the current population is correctly modified. Both steps will be discussed in more detail.

1. In the first step, atomic tuples are inserted into the tree population using the internal $\mathsf{Insert}$ operation. An atomic tuple is a tuple having only a root value. Such a tuple should only be inserted if there is no other tuple with this root value:

   **for each** $p \in f$
   **do**
     let $n = \mathsf{Node}(p)$
     **if** $\forall_{x \in \mathsf{Pop}(n)} [t(p) \neq x(n)]$
     **then** $\mathsf{Insert}(\mathsf{Pop}(n), t')$ where:
       $t'(n) = t(p)$
       $t'(g) = \emptyset$ (for all fact types $g$ with $\mathsf{Hook}(g) \in n$)
     **end**
   **end**

   **Example 6.1** *Consider the information structure from figure 2 and the tree representation from figure 3. The insertion of $\langle b_2, c_1, d_2 \rangle$ into $\mathsf{Pop}(g)$ was discussed in example 5.1. Now we show the necessary actions to be performed on the corresponding tree population.*

   *In this first step, we see that there is no tuple in the population of node $n_1$ with root value $b_2$ (section 4.3). As a consequence, we have to perform $\mathsf{Insert}(\mathsf{Pop}(n_1), t')$, where atomic tuple $t'$ is defined as follows:*

$$\begin{aligned} t'(n_1) &= b_2 \\ t'(n_2) &= \emptyset \\ t'(n_3, n_4) &= \emptyset \end{aligned}$$

2. Next, we consider the second step of the mapping process of $\mathsf{Insert}(\mathsf{Pop}(f), t)$ from the conceptual to the internal level. After the first step, all values in $t$ will occur in the corresponding tree population. However, they have to be interconnected. In the second step this interconnection is established using the internal $\mathsf{Modify}$ operation.

   Let $n = \mathsf{Node}(\mathsf{Hook}(f))$ be the upper node involving fact type $f$ and let $t'$ be a tuple in $\mathsf{Pop}(n)$ with the correct root value: $t'(n) = t(\mathsf{Hook}(f))$. Then, we have to perform the operation $\mathsf{Modify}(\mathsf{Pop}(n), t', \mathsf{m})$, where $\mathsf{m}$ is a modification function with domain consisting of all children of node $n$ dealing with fact type $f$ (i.e. $\mathsf{Dom}(\mathsf{m}) = \{\mathsf{Ext}(f)\}$). If $\mathsf{Ext}(f) = \{n_1, ..., n_l\}$, modification function $\mathsf{m}$ is defined as follows:

$$\mathsf{m}(f) = t'(f) \cup \varepsilon(f, t)$$

   where $\varepsilon(f, t)$ denotes the existing tuples in $\mathsf{Ext}(f)$ with a root value occurring in tuple $t$: $\varepsilon(f, t) = \{ \langle t_1, ..., t_l \rangle \mid \forall_{1 \leq i \leq l} [t_i \in \mathsf{Pop}(n_i) \wedge t_i(n_i) = t(\mathsf{Anchor}(n_i))] \}$

**Example 6.2** *In the second step, the new (atomic) tuple $t'$ from example 6.1 is modified by* $\mathsf{Modify}(\mathsf{Pop}(n_1), t', \mathsf{m})$*, with the following modification function* $\mathsf{m}$*:*

$$\begin{aligned} \mathsf{Dom}(\mathsf{m}) &= \{(n_3, n_4)\} \\ \mathsf{m}(n_3, n_4) &= \{\langle t_3, t_6 \rangle\} \end{aligned}$$

*where tuples $t_3$ and $t_6$ are defined as in example 4.3. The result is shown in figure 12. This example shows that an update of a fact type on the conceptual level may cause updates of several fact types on the internal level. For instance, the population of column $h$ in the nested table has been changed, while on the conceptual level it was not.*

| $n_1$ | $f$ | $g$ | | |
|---|---|---|---|---|
| | $n_2$ | $n_3$ | $n_4$ | $h$ |
| | | | | $n_5$ |

| | | | | |
|---|---|---|---|---|
| $b_1$ | $\boxed{a_1}$ | $c_1$ | $d_1$ | $\boxed{e_1}$ |
| | | $c_2$ | $d_1$ | $\boxed{e_1}$ |
| | | $c_2$ | $d_2$ | $\boxed{e_1}$ |
| $b_2$ | | $c_1$ | $d_2$ | $\boxed{e_1}$ |

Figure 12: Result of transformation

As a result of their formal nature, the examples in this section may seem rather artificial. Therefore we give a more practical example in terms of the Project Database shown in figure 4.

**Example 6.3** *This example is similar to examples 6.1 and 6.2. Suppose we record the fact that person 'Smith' is involved in project 'database design' for the duration of '2 days per week'. Then what happens on the internal level?*

*In the first step, the algorithm detects the absence of project 'database design'. As a consequence an atomic tuple with root value 'database design' will be inserted in the project column. In the second step, the values 'database design', 'Smith' and '2 days per week' are connected with each other.*

*Note that the second step properly deals with the nested nature of the internal representation, since the resulting tuple with root value 'database design' will also contain the department of person 'Smith'.*

Next we will shortly discuss the computational complexity of the insert operation. Suppose we insert a tuple $t$ into the population of fact type $f$. We express the complexity of this operation in terms of the number of internal updates needed.

Let $C_i$ and $C_m$ be the average cost of an internal insertion and modification respectively. In the first step, the number of insertions will not exceed the number of predicators in $f$, while in the second step there is exactly one modification. As a consequence, the complexity of the insert operation is given by:

$$C(\mathsf{Insert}(\mathsf{Pop}(f), t)) \leq |f| * C_i + C_m$$

Note that this is only a global estimation. For instance, in both steps of the transformation process some search must be done, in order to determine the precise actions to be performed. We will not elaborate on this further in this paper.

## 6.2 Transformation of deletion

In this section we consider the transformation of deletions from the conceptual to the internal level. Let $f$ be a fact type. Suppose we want to delete a tuple $t$ from the population of fact type $f$. This deletion is performed by $\mathsf{Delete}(\mathsf{Pop}(f), t)$. Let $T$ be a tree representation in which $f$ is represented internally. We describe how the corresponding tree population must be changed. This will (again) happen in two steps:

```
proc TransformDeletion (f : Fact Type; t : Tuple);
    Disconnect (f, t);
    DeleteAtomicTuples (f, t)
end
```

First the values in tuple $t$ must be disconnected such that the current population is correctly modified. Then the unnecessary atomic tuples can be deleted. Both steps will be discussed in more detail.

1. In the first step, values in the tree population are disconnected, according to the tuple to be deleted from $\mathsf{Pop}(f)$. This disconnection will be established using the internal $\mathsf{Modify}$ operation. Obviously the disconnection is described by the inverse of the second step in the previous section.

   Let $n = \mathsf{Node}(\mathsf{Hook}(f))$ be the upper node involving fact type $f$ and let $t'$ e a tuple in $\mathsf{Pop}(n)$ with the correct root value: $t'(n) = t(\mathsf{Hook}(f))$. Then, we have to perform the operation $\mathsf{Modify}(\mathsf{Pop}(n), t', \mathsf{m})$, where $\mathsf{m}$ is a modification function with domain $\mathsf{Dom}(\mathsf{m}) = \{\mathsf{Ext}(f)\}$. If $\mathsf{Ext}(f) = \{n_1, ..., n_l\}$, the modification function $\mathsf{m}$ is defined as follows:

   $$\mathsf{m}(f) = t'(f) - \varepsilon(f, t)$$

2. In the second step, the unnecessary tuples will be deleted from the tree population, using the internal $\mathsf{Delete}$ operation. As a consequence, all atomic tuples for which there is no reference from the father node will be deleted:

   ```
   for each p ∈ f
   do
       let n = Node(p)
       let t' ∈ Pop(n) such that t'(n) = t(p)
       if ¬∃_{t* ∈ Pop(Father(n))} [t' ∈ t*(n)] and ∀_{g ∈ F} [n = Node(Hook(g)) ⇒ t'(g) = ∅]
       then Delete(Pop(n), t')
       end
   end
   ```

   Note that this second step is the inverse of the first step in the previous section.

Next we will shortly discuss the computational complexity of the delete operation. This will be similar to the insert operation. Let $C_d$ and $C_m$ be the avarage cost of an internal deletion and modification respectively. Then the complexity of the delete operation has the following upper-bound:

$$C(\mathsf{Delete}(\mathsf{Pop}(f), t)) \leq |f| * C_d + C_m$$

## 6.3 Transformation of modification

In this section we consider the transformation of modifications from the conceptual to the internal level. Let $f$ be a fact type. Suppose we want to modify a tuple $t$ in the population of fact type $f$, according to modification function $\mathsf{m}$. This modification is performed by $\mathsf{Modify}(\mathsf{Pop}(f), t, \mathsf{m})$. Let $T$ be a tree representation in which $f$ is represented internally. The population of $T$ must be changed as follows:

**proc** TransformModification ($f$ : Fact Type; $t, \mathsf{m}$ : Tuple);
  TransformDeletion ($f, t$);
  TransformInsertion ($f, M(t, \mathsf{m})$)
**end**

First the old tuple $t$ is deleted and then the modified tuple $M(t, \mathsf{m})$ is inserted. In section 5.3 it was mentioned that this approach will in general be inefficient, since it may cause a deletion of atomic tuples, followed by an insertion of the same atomic tuples. A more efficient approach is obtained by restricting tuple $t$ to the domain of $\mathsf{m}$ before it is deleted, and by interleaving the execution of TransformDeletion and TransformInsertion.

This restriction and interleaving of deletion and insertion is done as follows. First tuple $t$ is partly disconnected (corresponding to $\mathsf{Dom}(\mathsf{m})$). Then the necessary atomic tuples are inserted and connected. Finally, the unnecessary atomic tuples can be deleted.

### 6.4 Query transformation

Although we focus on update operations and their transformation, a general strategy for retrieval operations is discussed in this section. We consider the following elementary search action for information structures:

> Let $p$ and $q$ be predicators in the same fact type $f$. Then, for a given value $x \in \mathsf{Pop}(\mathsf{Base}(p))$ occurring in $\mathsf{Pop}(f)$ via predicator $p$, find the values from $\mathsf{Pop}(\mathsf{Base}(q))$ being associated with $x$ via $p$ and $q$.

This elementary search action is an example of a low-level action, to be used when queries on information structures are performed. They can be applied during the evaluation of path expressions through information structures, occurring in conceptual languages like RIDL [14] and LISA-D [17].

Next we consider the transformation of the above search action for information structures into search actions for tree representations of information structures. Let $m$ be the node containing predicator $p$, and let $n$ be the node containing predicator $q$. Then the search action is from $m$ to $n$. Following wellformedness condition 5 from section 3.2, the following cases are distinguished:

1. If there is an edge $\langle n, m \rangle$, the search action is directed downwards. Then for the unique tuple $t \in \mathsf{Pop}(m)$ with root value $x$, those tuples in $\mathsf{Pop}(n)$ that are contained in $t$ are yielded.

2. If there is an edge $\langle m, n \rangle$, the search action is directed upwards. Then for the unique tuple $t \in \mathsf{Pop}(m)$ with root value $x$, those tuples in $\mathsf{Pop}(n)$ containing $t$ are yielded.

3. If there is a node $z$ with (equally labelled) incoming edges $\langle m, z \rangle$ and $\langle n, z \rangle$, the search action occurs between siblings. Let $t$ be the unique tuple in $\mathsf{Pop}(m)$ with root value $x$ and let $\Gamma \subseteq \mathsf{Pop}(z)$ be the set of tuples containing $t$. Then those tuples in $\mathsf{Pop}(n)$ that are contained in $\Gamma$ are yielded.

Obviously a path through an information structure will result in a path through the corresponding tree representation. The case distinction given above shows how the latter path can be derived.

# 7 Conclusions

In this paper we considered the transformation of databases from the conceptual to the internal level. We used an object-role data model on the conceptual level, along with a formalism for internal representations. The conceptual model, the internal representations and the relationship between these levels were defined in terms of one central notion, called the *predicator*. It was shown that this approach can be used appropriately for the transformation of database populations and operations.

The use of predicators seems to have several restrictions. Firstly, the ER approach ([12]) does not fit as neatly as NIAM ([21], [24]) and the Binary Relationship Model ([20], [27]). Secondly, the definition of complex object types is not straightforward. This problem was considered in [18] and it was shown that predicators can be used to define very complex object types, involving e.g. generalization, set types, sequence types and hierarchical decompostion. For more details we refer to [18].

We believe that our tree representations provide a useful intermediate language between existing conceptual and internal data modelling techniques. However, the approach still has some restrictions. A natural extension of the framework in figure 1 would be the transformation of database constraints. Also it will be clear that some additional background in database storage structures (such as index structures and physical storage structures) has to be incorporated.

The approach is currently being implemented in a prototype database optimizer. This optimizer performs a probabilistic search through the space of all possible internal representations of a given conceptual model. For more details, we refer to [6], [10], [5] and [7].

**Acknowledgement**

We would like to thank Th.P. van der Weide and E.D. Falkenberg of the Department of Information Systems, University of Nijmegen, The Netherlands, for their constructive discussions and comments on this research. The remarks of the anonymous referees resulted in many improvements.

# References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data structures and algorithms.* Addison-Wesley, Reading, Massachusetts, 1983.

[2] A. Amikam. On the automatic generation of optimal internal schemata. *Information Systems*, 10(1):37–45, 1985.

[3] A.D. Atri and D. Sacca. Equivalence and mapping of database schemes. In *Proceedings of the Tenth International Conference on Very Large Data Bases*, pages 187–195, 1984.

[4] D.S. Batory and C.C. Gotlieb. A unifying model of physical databases. *ACM Transactions on Database Systems*, 7(4):509–539, December 1982.

[5] P. van Bommel. Experiences with EDO: an Evolutionary Database Optimizer. *Data & Knowledge Engineering.* To appear.

[6] P. van Bommel. A Randomised Schema Mutator for Evolutionary Database Optimisation. *The Australian Computer Journal*, 25(2):61–69, May 1993.

[7] P. van Bommel. Database Design Modifications based on Conceptual Modelling. In H. Kangassalo and H. Jaakkola, editors, *Information Modelling and Knowledge Bases V*, Amsterdam, The Netherlands, 1993. IOS Press. In Press.

[8] P. van Bommel, A.H.M. ter Hofstede, and Th.P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5):471–495, October 1991.

[9] P. van Bommel and Th.P. van der Weide. Reducing the search space for conceptual schema transformation. *Data & Knowledge Engineering*, 8:269–292, 1992.

[10] P. van Bommel and Th.P. van der Weide. Towards Database Optimization by Evolution. In A.K. Majumdar and N. Prakash, editors, *Proceedings of the International Conference on Information Systems and Management of Data (CISMOD 92)*, pages 273–287, Bangalore, India, July 1992.

[11] A. Borgida. Features of Languages for the Development of Information Systems at the Conceptual Level. *IEEE Software*, pages 63–72, January 1985.

[12] P.P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[13] L.S. Colby. A recursive algebra for nested relations. *Information Systems*, 15(5):567–582, 1990.

[14] O.M.F. De Troyer, R. Meersman, and F. Ponsaert. RIDL User Guide. Research report, International Centre for Information Analysis Services, Control Data Belgium, Inc., Brussels, Belgium, 1984.

[15] J.J. van Griethuysen, editor. *Concepts and Terminology for the Conceptual Schema and the Information Base*. Publ. nr. ISO/TC97/SC5-N695, 1982.

[16] T.A. Halpin. A Fact-Oriented Approach to Schema Transformation. In B. Thalheim, J. Demetrovics, and H.-D. Gerhardt, editors, *MFDBS 91*, volume 495 of *Lecture Notes in Computer Science*, pages 342–356, Rostock, Germany, 1991. Springer-Verlag.

[17] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, 1993.

[18] A.H.M. ter Hofstede and Th.P. van der Weide. Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100, February 1993.

[19] E. Horowitz. *Programming Languages*. Springer-Verlag, 1983.

[20] I. Kobayashi. Classification and transformations of binary relationship relation schemata. *Information Systems*, 11(2):109–122, 1986.

[21] C.M.R. Leung and G.M. Nijssen. Relational database design using the NIAM conceptual schema. *Information Systems*, 13(2):219–227, 1988.

[22] P Lyngbaek and V. Vianu. Mapping a semantic database model to the relational model. In *SIGMOD Record*, pages 132–142, 1987.

[23] E. Mendelson. *Introduction to Mathematical Logic*. Wadsworth and Brooks, 1987.

[24] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice-Hall, Sydney, Australia, 1989.

[25] M.A. Roth, H.F. Korth, and A. Silberschatz. Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems*, 13(4):389–417, December 1988.

[26] H.J. Schek and M.H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[27] P. Shoval and S. Zohn. Binary-Relationship integration methodology. *Data & Knowledge Engineering*, 6(3):225–250, 1991.

[28] J.D. Ullman. *Principles of Database and Knowledge-base Systems*, volume I. Computer Science Press, Rockville, Maryland, 1989.

[29] Th.P. van der Weide, A.H.M. ter Hofstede, and P. van Bommel. Uniquest: Determining the Semantics of Complex Uniqueness Constraints. *The Computer Journal*, 35(2):148–156, April 1992.