

# DEDUCTIVE INFORMATION RETRIEVAL BASED ON CLASSIFICATIONS

Kalervo Järvelin<sup>+</sup> and Timo Niemi<sup>#</sup>

<sup>#</sup>Dept. of Computer Science and

<sup>+</sup>Dept. of Information Studies

University of Tampere

P.O.Box 607

SF-33101 TAMPERE, Finland

## Abstract

Modern fact databases contain abundant data classified through several classifications. Typically users must consult these classifications in separate manuals or files thus making their effective use difficult. Contemporary database systems do little to support deductive use of classifications. In this paper we show how deductive data management techniques can be applied to the utilization of data value classifications. Computation of transitive class relationships is here of primary importance. We define a representation of classifications which supports transitive computation and present an operation-oriented deductive query language tailored for classification-based deductive information retrieval. The operations of this language are on the same abstraction level as relational algebra operations and can be integrated with these to form a powerful and flexible query language for deductive information retrieval. We define the integration of the operations and demonstrate the usefulness of the language in terms of several sample queries.

**Key-words** : fact databases, information retrieval, classifications, deduction.

## 1. Introduction

Fact databases provide data for science and technology (e.g. properties of chemical compounds), business and trade (e.g. data on marketing, banking or shares), and social science and governmental activities (e.g. various statistics) (Chen & Herson, 1984 ; Wolski & al., 1990). In 1989, 24 % of the 4062 public online databases were classified as fact databases or textual-numeric databases (Tenopir & Ro, 1990). Many fact databases contain primarily or solely numeric data. Essential characteristics of fact databases are their *structuredness*, i.e. the data are organized as named data items whose values are stored in separate fields, *factuality*, i.e. the data are facts which are represented as data item values as opposed to narrative text, and *exactness of matching*, i.e. a unique correct query expression which retrieves all (and only) relevant data can be determined prior to query execution.

Modern fact database environments have several characteristics making them difficult and unsatisfactory for end-users and information specialists. Such characteristics include differing data models, data structures, attribute naming conventions, units of measurement or naming of data values, composition of data as attributes, technical representation of data, abstraction levels of data, etc. Many problems must therefore be solved in order to bring data from different sources together for effective utilization. (Chen & Hernon, 1984; Järvelin, 1988; Järvelin & Niemi, 1990)

Among the naming problems of data item values, the problems of parallel classifications are prominent. In fact databases data item values are often classified on the basis of external classifications. For example, in statistical databases related to economic activities, data on product types, chemical compounds, labour or geographic areas are often classified by more and less widely recognized classifications (including those provided by the OECD) (Wolski & al, 1990). Very often the most basic stored data are classified multi-dimensionally through several classifications, e.g. as chemical compound category, health-hazard category, geographically and chronologically. These classifications must often be consulted in separate manuals or files. Thus their effective use is impaired. Deductive use of classifications, in particular, is badly supported by contemporary fact database systems. This problem area calls for the integration and application of information retrieval (IR) and deductive data management techniques.

Traditional IR systems for document and reference databases provide some methods for the utilization of classifications (or thesauri). These include online thesauri (or classifications) and hierarchical term (or class) expansion (Harter, 1986; Lancaster, 1986). Online thesauri allow online consultation of thesauri and support manual term selection for queries. Hierarchical term expansion means expanding a term to a disjunction of its hierarchically narrower terms through the levels of the hierarchy. The user need not know and/or write all the narrower terms. The terms describing a document form a set stored in an index term field. Each term represents document content in an equal (or sometimes weighted) way. Except for index term fields, other attributes describing document content are not classified by external thesauri (or classifications). Terms at any hierarchical level may be used for document content description.

Traditional IR databases and methods do not allow deduction. This is because they employ a single thesaurus (if any) for a single attribute (if any), and also because they provide only one unidirectional operation (the hierarchical expansion toward narrower terms) for automatic thesaurus manipulation. Features like the ZOOM (ESA, n.d.) or automatic query expansion (e.g. Peat & Willett, 1991) available in some operational or experimental systems make no

difference as they are based on attribute value frequencies (ZOOM) or statistically related term occurrences (query expansion), neither of which allows deduction.

Fact databases differ in certain aspects. Limited forms of online thesauri (or classifications) and hierarchical term expansion are sometimes available (Wolski & al, 1990). Most importantly, the entities represented in fact databases are usually described through several attributes (in distinct fields), each of which may be associated with a classification or a thesaurus (Hoppe & al., 1990). Moreover, often only the terms (classes) at the lowest levels of the hierarchies are used to represent the entities -- higher levels are used for data aggregation. The same basic attribute values may also be classified through more than one classification hierarchy (e.g. by chemical structure and health-hazard classifications).

Current fact database systems allow only simple utilization of multiple hierarchical classifications. For example, data can be aggregated in multidimensional classes formed by chemical compound categories, geographic areas and time intervals (months and years) (Wolski & al., 1990). Data aggregation is an important classification-related feature in fact databases. However, the authors have addressed it in earlier publications (Järvelin & Niemi, 1991a-b ; Järvelin & Niemi, 1992).

Data aggregation, however, is only one way to utilize classifications in fact databases. In the present article we shall study deductive queries which constitute another important application area of classifications. Deductive queries are based on transitive class relationships and may traverse hierarchies toward subclasses (i.e. "top-down") in one classification and toward superclasses ("bottom-up") in another. For example, queries like "What types of health-hazards are associated with the CFC-family of chemicals ?" and "Which districts are affected by allergenic air pollutants ?" are highly relevant in many conceivable fact database environments. The former query would entail finding subclasses in the classification of (CFC-) chemicals and then their superclasses at the health-hazard type level in the classification of health-hazards. The latter query would require going down the classification of (allergenic) health-hazards and up the classification of geographic areas (from specific sites to districts). Traditional retrieval methods of online fact database systems do not allow such a capability. Neither do they allow integration of such queries with fact retrieval (e.g. "Which cities suffer from nitrogen monoxide pollution more than London ?"). Thus these methods and systems are clearly insufficient. In practise, such query capabilities are often needed and advanced IR systems should provide them.

Conventional query languages for structured databases are suitable for the multi-attribute description of entities but provide hardly any support for the utilization of classifications. This is due to the lack of a recursion mechanism in the languages (e.g. Ullman, 1988; Zloof, 1975) ; without it, deductive utilization of classifications is impossible because only the immediate

subclasses (or one hierarchical level) can be identified through conventional operations. It is generally accepted that conventional query languages cannot utilize hierarchical relationships transitively.

Deductive information retrieval techniques are needed due to the limitations of IR methods, fact database systems and conventional query languages. Deductive information retrieval refers to seamless and flexible integration of transitive computation (based on classifications) and conventional retrieval methods. This makes it possible for deductive and conventional operations to alternate and also for deductive operations to traverse multiple classifications both upwards and downwards in a single powerful query expression (Järvelin & Niemi, 1991). Much effort has been devoted in database research to developing deductive capabilities in data management (e.g. Bancilhon & Ramakrishnan, 1988; Jagadish & Agrawal, 1987; McLean & Weise, 1991; Parsaye & al., 1989; Ullman, 1989). However, so far little attention has been devoted to the application of suitable deductive techniques in the context of IR problems where classifications and thesauri are already available.

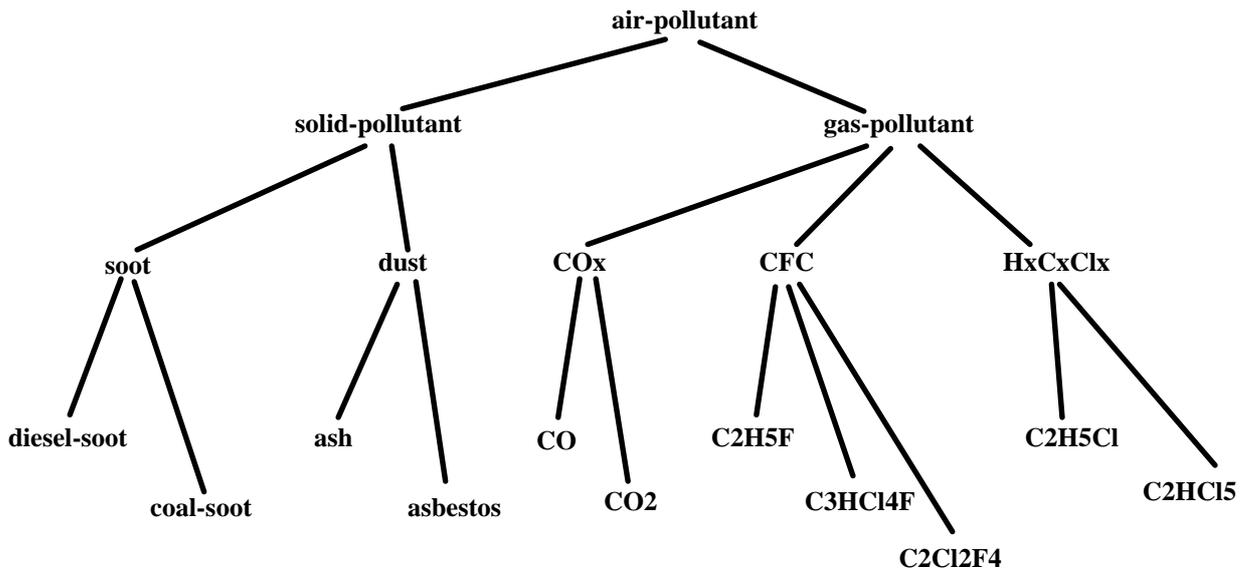
This paper presents operations which allow the deductive use of classifications and shows how these operations can be integrated with relational algebra (RA) to yield a powerful deductive information retrieval language. This operation-oriented query language provides essential support in overcoming the limitations and retrieval problems due to classified information. Several sample queries are used to illustrate this support. They exhibit novel features for fact retrieval which are possible only by integrating deductive techniques with the more traditional classifications of IR. We have defined the query language to make it relatively easy and transparent. In particular, it does not require that the users master recursion or recursive definition in some programming language, as is typically required for using contemporary deductive databases.

Our general approach to deductive processing (Niemi & Järvelin, 1992a-c) is here adapted to classification-based deductive information retrieval. Thus we define all data organization and processing related to classifications using classification terminology. We shall introduce a set of deductive operations tailored for the processing of classifications and then show how the classifications should be represented for these operations. In hierarchical classifications there are typically many immediate class-subclass relationships. In our approach these are stored explicitly, and from them transitive class relationships can be derived - e.g. in hierarchical class expansion, all subclasses, sub-subclasses, etc. are identified. Our operations are dominated by those for transitive class relationship computation.

These deductive operations and their integration with RA in Prolog have been implemented in a workstation environment. We shall focus on the operations, their integration and the expressive power of our system but by-pass the details of user interfaces (e.g. visualization of







**Fig. 2.1 (a)** Partial tree-representation of the classification air-pollutants

pollutant-types	
air-pollutant	solid-pollutant
air-pollutant	gas-pollutant

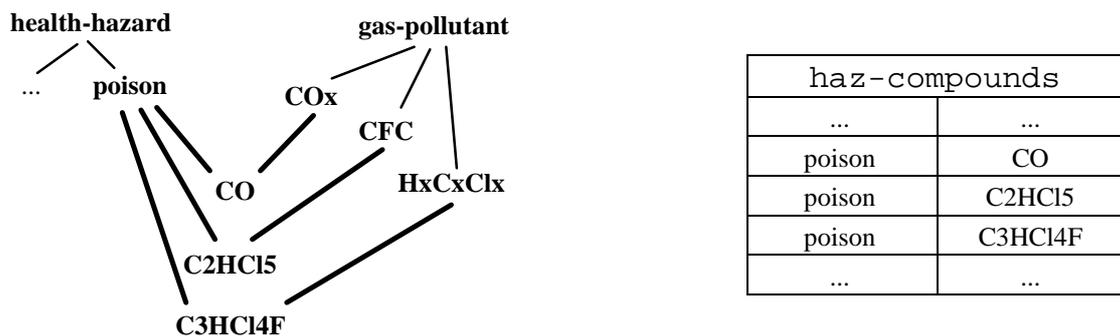
  

pollutant-families	
solid-pollutant	soot
solid-pollutant	dust
gas-pollutant	CFC
gas-pollutant	COx
...	...

pollutants	
soot	diesel-soot
soot	coal-soot
COx	CO
COx	CO2
CFC	C2H5F
CFC	C3HC14F
CFC	C2C12F4
HxCxClx	C2HCl5
...	...

**Fig. 2.1 (b)** Partial binary relations for the classification air-pollutants



**Fig. 2.1 (c)** Representation of poly-hierarchical relationships by binary relations

Each classification is represented so that each pair of immediate hierarchical levels (i.e. each basic classification) forms a binary relation. The binary relations for the classification air-pollutants are thus as outlined in Fig. 2.1 (b). Each class level forms its own (named) clas-

sification domain. The root classification domain in the classification `air-pollutants` has the name *air-pollutant* and contains a single value `air-pollutant`, i.e.  $air-pollutant = \{air-pollutant\}$ . The second classification domain is  $poll-type = \{solid-pollutant, gas-pollutant\}$ . The third classification domain is  $poll-family = \{soot, dust, CFC, COx, \dots\}$  and the leaf classification domain is  $pollutant = \{diesel-soot, coal-soot, CO, CO_2, \dots\}$ . The top-most binary relation (and basic classification) for the hierarchical classification `air-pollutants` is called `pollutant-types`, which is a binary relation in the Cartesian product  $air-pollutant \times poll-type$ . Analogously, the second binary relation, `pollutant-families`, is a relation in the Cartesian product  $poll-type \times poll-family$ , and the third, `pollutants`, is a relation in the Cartesian product  $poll-family \times pollutant$ .

Fig. 2.1 also exemplifies the representation of poly-hierarchical class relationships by several binary relations. Some poly-hierarchical class relationships are shown in the tree-representation on the left in Fig. 2.1 (c). The binary relation `haz-compounds` on the right represents CO, C<sub>2</sub>HCl<sub>5</sub>, and C<sub>3</sub>HCl<sub>4</sub>F as poisons. The binary relations `pollutant-families`, `pollutants` and `haz-compounds` (Fig. 2.1 (b) & (c)) together represent, among others, the poly-hierarchical class relationships of carbon monoxide as a poison and as a gas-pollutant. Binary relations themselves do not require that the basic classifications of two classification principles (e.g. `pollutant-types`, `health-hazard`) should be stored in separate relations. From the representation point-of-view they could be united, i.e. the relations `pollutants` and `haz-compounds` could be united. However, it is our design choice that classifications of different classification principles are stored separately : this allows control when needed over the direction of deduction. In other words, it is possible to find the superclasses of, say C<sub>3</sub>HCl<sub>4</sub>F, either in the health-hazard direction, the pollutant-type direction or both directions. The union of the two relations would always yield superclasses simultaneously in both directions, without any syntactic cues for their semantic differences.

For each sample classification Appendix I also gives the names of their associated classification domains related to the class levels, and, further, the names and signatures of the binary relations representing them. It is straightforward to translate the classification levels to the respective classification domains and the hierarchical divisions to binary relations (basic classifications). We do not therefore list here explicitly the classification domains or the binary relations.

The classifications are augmented by thesauri, which give for each relevant entry term its class, its classification domain, definition, scope, synonyms, translations and related terms as appropriate (broader and narrower terms are given by the classifications). The domain description relates each term (and class) to a particular classification domain. The attributes of the relations (see below) are defined over given domains. It is therefore possible to find for

any classified relational attribute value its synonyms and other information provided by the thesaurus. Contrary to Hoppe & al. (1990), we prefer to associate the thesauri with classification domains rather than relational attributes, because the same domain may be shared by several attributes and the attributes are more transient than the domains. The thesauri can be represented to users as "thesaurus" relations. This paper, however, does not deal with thesaurus content and its non-deductive use as these are quite obvious in IR and have already been analyzed in fact database environments (Hoppe & al., 1990). The relational database below includes only a simple dictionary relation.

## 2.2. Sample Relations

The relations of the sample database RDB1 are outlined in Figure 2.2. The schema components are given in the first three rows of the relations: the relation name in the first row, the attributes in the second row and their domains in the third row. The following rows illustrate the relation instance. The attributes DMinTmp, DMaxTmp and DAvTmp contain the daily minimum, maximum and average temperature for the sensor station (TStat) and the date. The attributes Pollutant, CritCont and Recipient contain data on the critical air content of various pollutants for various recipients. The attributes PollName and Class give common names and class names of various pollutants. The attributes Ptant and PMesment contain the pollutant name and average pollutant air content data for the sensor station (PStat) and date. In these relations, the attributes TStat and PStat are associated with the geographic and sensor station type classifications, and the the attributes Pollutant, Class and Ptant are associated with the pollutant, health hazard and greenhouse gas classifications.

TEMPERATURE							
DMinTmp	DMaxTmp	DAvTmp	TStat	TDay	TMonth	TYear	...
°F	°F	°F	gb-stat-name	day-int	month-ch	year-int	...
30	38	33	london1	12	jan	91	...
35	44	40	london2	15	jan	91	...
...	...	...	...	...	...	...	...

CRITICALCONTENT			
Pollutant	CritCont	Recipient	...
pollutant	Oz-in-ft3	string	...
SO2	0.0000003	humans	...
SO2	0.0000004	forests	...
NO2	0.0000002	birds	...
...	...	...	...

DICTIONARY		
PollName	Class	...
string	pollutant	...
sulphur dioxide	SO2	...
Chlor-Fluor-Carbons	CFC	...
DDT	C14H9Cl5	...
...	...	...

POLLUTANTS						
Ptant	PMesment	PStat	PDay	PMonth	PYear	...
pollutant	Oz-in-ft3	gb-stat-name	day-int	month-ch	year-int	...
SO2	0.0000005	london1	12	jan	91	...
NO2	0.0000001	london2	15	jan	91	...
...	...	...	...	...	...	...

Fig. 2.2. Relations in the sample database RDB1

### 3. Formal Representation of Classifications

Chapter 2 described the way our approach represents information. Formal representation of relations was considered in (Niemi & Järvelin, 1985). We now consider the formal representation of classifications. A classification base basically consists of classes. We denote the set of classes by **CLASS**. A *classification base* CB is formally a set of basic classifications  $BC_1, BC_2, \dots, BC_n$ , i.e.  $CB = \{BC_1, BC_2, \dots, BC_n\}$ . Our sample classification base CB1 is, on the basis of Appendix 1, the set  $CB1 = \{\text{pollutant-types}, \text{pollutant-families}, \text{pollutants}, \text{hh-types}, \text{haz-compounds}, \text{gh-gases}, \text{station-types}, \dots\}$ . Each basic classification  $BC_i, 1 \leq i \leq n$ , is a binary relation in  $CLASS \times CLASS$ . Each basic classification  $BC_i, 1 \leq i \leq n$ , is therefore represented formally as  $\{ \langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle, \dots, \langle a_m, b_m \rangle, \}$ , where  $\forall j \in \{1, \dots, m\} : a_j \in CLASS \wedge b_j \in CLASS$ . Each element  $\langle a_j, b_j \rangle, j \in \{1, \dots, m\}$ , expresses intuitively a hierarchical relationship between the classes  $a_j$  and  $b_j$  so that  $a_j$  is interpreted as an *immediate superclass* of  $b_j$ , and  $b_j$  as an *immediate subclass* of  $a_j$ . For example, CFC is an immediate superclass of C2H5F in the sample basic classification pollutants.

In each basic classification  $BC = \{ \langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle, \dots, \langle a_m, b_m \rangle, \}, 1 \leq j \leq m$ , the *superclass domain*  $UD = \{a_1, a_2, \dots, a_m\}$  and the *subclass domain*  $LD = \{b_1, b_2, \dots, b_m\}$  form the classification domains. The nature of classifications requires that in each basic classification the superclass and subclass domains are disjoint, i.e.  $UD \cap LD = \emptyset$ . Note that, given the su-

per- and subclass domains UD and LD, the basic classification BC is a relation in  $UD \times LD$ , i.e.  $UD \times LD \supseteq BC$ .

Multi-level hierarchical classifications are represented by a collection of basic classifications. In such a collection the classifications must be connected. The connectedness condition between classifications is given as follows :

*Definition 3.1* : Let  $a, b, c, d (\in \mathbf{CLASS})$  be any classes. Two basic classifications  $BC_1$  and  $BC_2$  are *connected*, denoted by  $conn(BC_1, BC_2)$ , if :

- 1•  $\{ b \mid \langle a, b \rangle \in BC_1 \} \cap \{ c \mid \langle c, d \rangle \in BC_2 \} \neq \emptyset$  or
- 2•  $\{ a \mid \langle a, b \rangle \in BC_1 \} \cap \{ d \mid \langle c, d \rangle \in BC_2 \} \neq \emptyset$ .

The definition states that a basic classification  $BC_1$  can be connected to a basic classification  $BC_2$  in two directions. The first condition expresses that  $BC_2$  contains subclasses of the subclass domain of  $BC_1$ . The second condition expresses that  $BC_2$  contains superclasses of the superclass domain of  $BC_1$ . In the former case  $BC_1$  is connected to  $BC_2$  in the subclass direction, and in the latter, in the superclass direction. Intuitively, the basic classifications are connected if the superclass domain of one matches the subclass domain of the other.

A multi-level hierarchical classification is represented as such a collection of basic classifications  $HC = \{BC_1, BC_2, \dots, BC_n\}$  where the basic classifications are connected in a unique way. Therefore there is exactly one order among the basic classifications so that they are connected pair-wise. In order to define this formally, we need the concept of permutation.

*Definition 3.2* : If  $S$  is any set of  $n$  elements (i.e.  $|S| = n$ ), then any tuple  $\langle s_1, s_2, \dots, s_n \rangle$  such that  $\cup_{i=1..n} \{s_i\} = S$  is a *permutation* of  $S$ . The set of all permutations of  $S$  is denoted by *permutations(S)*.

The definition says, in effect, that every distinct ordering of the elements of a set is a distinct permutation. For example, if  $S = \{a, b, c\}$ , then  $\langle a, b, c \rangle$  and  $\langle a, c, b \rangle$  are permutations of  $S$  and  $permutations(S) = \{\langle a, b, c \rangle, \langle a, c, b \rangle, \langle b, a, c \rangle, \langle b, c, a \rangle, \langle c, a, b \rangle, \langle c, b, a \rangle\}$ . Thus permutation imposes an order on the elements of a set. The representation of a hierarchical classification is now defined as a collection of basic classifications such that there is exactly one permutation of the basic classifications in which the basic classifications are pair-wise connected.

*Definition 3.3* : A collection of basic classifications  $HC$  represents a hierarchical classification, if :

- 1•  $\exists P = \langle BC_1, BC_2, \dots, BC_n \rangle \in permutations(HC) : \forall i \in \{1, \dots, n-1\} : conn(BC_i, BC_{i+1})$ ,
- 2•  $\forall P' \in permutations(HC) : P' \neq P \Rightarrow \exists i \in \{1, \dots, n-1\} : \neg conn(BC_i, BC_{i+1})$ .

The first condition establishes the permutation  $P$  where the basic classifications are pair-wise connected and the second condition requires that there is no other permutation with this property. For example, the collection of basic classifications  $AP = \{\text{pollutant-types}, \text{pollutant-families}, \text{pollutants}\}$  represents the hierarchical classification `air-pollutants` because its elements are pair-wise connected in the permutation  $\langle \text{pollutant-types}, \text{pollutant-families}, \text{pollutants} \rangle$ .

The organization of hierarchical classifications as collections of connected basic classifications has important advantages. Firstly, the user can limit deduction to the very classifications which are of interest. Secondly, he can limit deduction exactly to those consecutive levels of hierarchy which are relevant. For example, one might consider the class relationships only among the relations `pollutant-types` and `pollutant-families`. This would provide an overview of the classification. Thirdly, our approach supports poly-hierarchical classifications of the same basic classes. For example, all the classifications `air-pollutants`, `health-hazard`, and `greenhouse-gas` classify the same basic classes, the pollutants, but each on the basis of a different characteristic. These features are necessary for deductive IR (cf. the manual use of thesauri in traditional IR) but are not provided by current deductive databases which define transitive relationships in a single large binary relation where selective processing is hard.

Because the basic classifications are binary relations in  $\mathbf{CLASS} \times \mathbf{CLASS}$  we can combine hierarchically and poly-hierarchically related basic classifications for deduction simply by the union operation. In a union of connected basic classifications there are always *transitive* (or *indirect*) class relationships. Deduction is based on computing such indirect class relationships. The operations introduced in Chapter 4 allow deduction on transitive relationships.

For example, let  $A$  and  $B$  be the following basic classifications :  $A = \{\langle a, b \rangle, \langle a, c \rangle, \langle d, e \rangle\}$  and  $B = \{\langle c, f \rangle, \langle e, g \rangle\}$ . Now  $A$  and  $B$  are connected and their union is the binary relation  $\{\langle a, b \rangle, \langle a, c \rangle, \langle d, e \rangle, \langle c, f \rangle, \langle e, g \rangle\}$ . Note that in this relation there are some classes (e.g.  $c$  and  $e$ ) which appear both in the first component of one pair and in the second component of another. In a basic classification, no one class can be given in both components. Due to the pairs  $\langle a, c \rangle$  and  $\langle c, f \rangle$  also  $a$  and  $f$  are indirectly connected. There is a *transitive* relationship between the classes  $a$  and  $f$ . We call  $a$  an *indirect superclass* of  $f$  and  $f$  is an *indirect subclass* of  $a$ .

Because all hierarchical classifications in a classification base contain a finite number of classes there are always some classes which have no super- or subclasses. In any set of basic classifications the former are called *root* classes and the latter *leaf* classes. These classes have a special role because they are the ultimate starting and termination levels of deduction. We define root and leaf classes formally as follows.

*Definition 3.4* : Let  $CS = \{BC_1, BC_2, \dots, BC_n\}$  be any collection of basic classifications and  $BCS = \cup_{i=1, \dots, n} BC_i$  the union of the basic classifications. The set :

$$\{ \text{class} \mid \langle \text{class}, \text{class1} \rangle \in BCS \wedge \neg \exists \text{class2} \in \mathbf{CLASS} : \langle \text{class2}, \text{class} \rangle \in BCS \}$$

is called the root class set of the collection CS and the set

$$\{ \text{class} \mid \langle \text{class1}, \text{class} \rangle \in BCS \wedge \neg \exists \text{class2} \in \mathbf{CLASS} : \langle \text{class}, \text{class2} \rangle \in BCS \}$$

is called the leaf class set of the collection CS.

Consider, for example, the sample classifications A and B above and let  $CS = \{A, B\}$ . In CS, the root classes are  $\{a, d\}$  and the leaf classes are  $\{b, f, g\}$ . The operations introduced in Chapter 4 allow the the root and leaf classes in any set of basic classifications to be deduced.

#### 4. An Operation-Oriented Query Language for Classifications

We now introduce an operation-oriented language for deductive manipulation of multiple classifications. The operations are based on the representation of the classification base defined in Chapter 3. We propose the operation-oriented approach for the manipulation of classes. It provides the user with automatic class manipulation and a greater expressive power and control over the length and direction of deduction than does traditional IR. Traditional hierarchical term expansion in IR will be shown to be a special case in deduction based on classifications. The operation-oriented approach offers more benefit through its suitability for integration with traditional fact retrieval operations in a powerful language for deductive information retrieval (Chapter 5).

In order to achieve such benefits the deductive classification operations must be defined at a very high abstraction level. In other words, transitive computation related to classes must be invisible to the user. In conventional approaches to deduction this requires recursive definition (Niemi & Järvelin, 1992a-c). Our high-level operations do not require the user to master recursive definition. The user only has to master the following aspects when formulating queries in the classification base :

- (i) understand the meaning of those classes in the classification base which he uses ;
- (ii) know the classifications in the classification base and know which basic classifications are connected and which of them represent a multi-level hierarchical classification ;
- (iii) understand the semantics of the deductive operations.

Requirement (i) is a requirement common to all IR systems based on classes and thesaural or natural language terms. Requirement (ii) is partly a new requirement when compared to traditional IR systems, but increases the user's control over the length and direction of deduction. It is easy to see that the relationships of basic classifications can be visualized as a network of basic classification names. Grasping their relationships and choosing those of interest should therefore not be difficult. Requirement (iii) is the (unavoidable) cost of greater expressive

power and control over deduction. However, we believe strongly that the operations introduced in this chapter are easily understandable -- even intuitively.

Our deductive classification operations can be classified in several ways which reflect their formal properties and semantic affinities :

- (i) *direction* : *superclass-directed*, *subclass-directed* and *non-directed* operations ;
- (ii) *transitivity* : *transitive* and *non-transitive* operations ;
- (iii) *result-type* : *class-oriented* operations vs. *path-oriented* operations.

(i) Subclass-directed operations expand a given class or a set of classes toward the subclasses for manipulation, whereas the superclass-directed operations expand a given class or a set of classes towards the superclasses. Our approach contains many operations which are directional counterparts of each other -- the only difference between them being the direction of deduction. The traditional term expansion in IR is always directed towards the subclasses. Some operations, like the conventional set-operations, are non-directed.

(ii) Some operations (e.g. those for identifying the root classes in a collection of basic classifications) do not need the computation of transitive relationships. Traditional IR does not provide operations for identifying classification roots or leaves either in general or in dynamically constructed collections of basic classifications. Most operations in our approach are transitive operations which employ the transitive class relationships in collections of basic classifications.

(iii) Our class-oriented operations give as their results either individual classes or sets of classes. In this paper we shall focus on these operations. The path-oriented operations produce classification paths (or sets of paths)  $\langle c_1, c_2, \dots, c_n \rangle$  where for each  $i = 1 \dots n-1$ ,  $c_i$  is an immediate superclass of  $c_{i+1}$  in the collection of basic classifications considered. These operations can be used for finding class and subclass chains in hierarchical classifications. They are best suited to classification analysis and will not be considered further in this paper.

We shall introduce informally our transitive and non-transitive class-oriented operations in Section 4.1. The direction of the operations is reflected in the operation names. The names of superclass-directed operations contain the word "superclass" or "root" and the names of subclass-directed operations the word "subclass" or "leaf". These operations are here adapted to classification-based deductive information retrieval from our earlier general-purpose operations for recursive queries (Niemi & Järvelin, 1992a-c). In Section 4.2 we show how the operations can be combined in nested expressions to cater for more complex information needs. Section 4.2 also provides examples of more complex purely classification-based queries.

#### 4.1. Class-oriented deductive operations

*A. The non-transitive operations.* Non-transitive operations cover operations for finding e.g. the root and leaf classes and immediate class relationships in collections of basic classifications, and for ordinary set manipulation.

Our operations for the root and leaf classes are `ROOT_CLASSES(Scope)` and `LEAF_CLASSES(Scope)`, respectively. In the argument `Scope` the user gives the names of the basic classifications among which the root classes (or leaf classes) are computed. The operations are expressed as follows :

```
ROOT_CLASSES({BC1, BC2, ..., BCn})
LEAF_CLASSES({BC1, BC2, ..., BCn})
```

for the root classes (or leaf classes) in the scope of `{BC1, BC2, ..., BCn}`. The former is, obviously, superclass-directed and the latter subclass-directed. For example, the expression `LEAF_CLASSES({pollutant-types, pollutant-families})` yields the classes `{soot, dust, COx, NOx, Ozone, SOx, CFC, HxCxClx}` and the expression `ROOT_CLASSES({pollutant-families, pollutants, haz-compounds})` yields the classes `{solid-pollutant, gas-pollutant, allergene, carcinogene, poison}` (see Fig. 2.1 and Appendix I).

The operation `CLASSES(Scope)` is analogous to the above but non-directed and returns all different classes within the scope.

The operations `IM_SUBCLASSES(Class, Scope)` and `IM_SUPERCLASSES(Class, Scope)` find the immediate subclasses (or immediate superclasses) of a given class (the argument `Class`) in the given collection of basic classifications (the argument `Scope`). In other words, these operations give the adjacent lower or upper level related to a class in the collection of basic classifications corresponding to `scope`. User expressions for these operations are as follows, when `class` denotes a class and the scope consists of the basic classifications `{BC1, BC2, ..., BCn}` :

```
IM_SUBCLASSES(class, {BC1, BC2, ..., BCn})
IM_SUPERCLASSES(class, {BC1, BC2, ..., BCn}).
```

For example, the expression `IM_SUBCLASSES(great-britain, {gb-regions, gb-sub-regions})` gives the result `{england, wales, scotland, northern-ireland}`, while the expression `IM_SUPERCLASSES(C3HCl4F, {pollutants, pollutant-families, haz-compounds, hh-types})` gives the result `{CFC, poison}`. Note that this result remains the same in the scope `{pollutants, haz-compounds}`. These examples give superclasses from two distinct classifications, i.e. they are an examples of the utilization of poly-hierarchical

relationships. These operations correspond to the narrower term and broader term look-ups for manual term selection in online thesauri.

The usual (non-directed) set operations `SET_INTERSECTION(set1, set2)`, `SET_UNION(set1, set2)`, and `SET_DIFFERENCE(set1, set2)` can also be applied to sets consisting of classes. For example, the expression `SET_INTERSECTION({C2H5F, C3HC14F, C2C14F2, C2C12F4}, {CO2, C2C14F2, C2C12F4})` yields the class set `{C2C14F2, C2C12F4}`.

*B. The transitive operations.* The basic transitive class-oriented operations are the operations `SUBCLASSES(Class, Scope)` and `SUPERCLASSES(Class, Scope)` which give all subclasses or superclasses of a class (the argument `Class`) in a collection of basic classifications (the argument `Scope`). The subclasses of a class comprise all its direct and indirect subclasses within `Scope`. Analogically, the superclasses of a class comprise all its direct and indirect superclasses in the collection of basic classifications corresponding to `Scope`.

For example, the expression `SUBCLASSES(great-britain, {gb-regions, gb-sub-regions})` gives the result `{england, north-england, mid-england, east-england, south-england, wales, north-wales, south-wales, scotland, east-scotland, west-scotland, northern-ireland, northern-ireland1}`, while the expression `SUPERCLASSES(C3HC14F, {pollutants, pollutant-families, haz-compounds, hh-types})` gives the result `{CFC, gas-pollutant, air-pollutant, poison, health-hazard}`. Here, too, the latter example utilizes poly-hierarchies by finding superclasses from two distinct classifications. The classes `CFC`, `gas-pollutant` and `air-pollutant` are the superclasses of `C3HC14F` in one direction and the classes `poison` and `health-hazard` in the other. Note that the traditional term expansion in IR corresponds to the expression `SUBCLASSES(term, Scope)` where the scope consists of all basic classifications representing a single hierarchical classification.

These basic transitive operations are generalized from individual class processing to processing sets of classes as follows. The operations `UNION_OF_SUBCLASSES(Class-set, Scope)` and `UNION_OF_SUPERCLASSES(Class-set, Scope)` compute all distinct subclasses or superclasses of a given set of classes (the argument `Class-set`) in the collection of basic classifications (the argument `Scope`). The operations `INTERSECTION_OF_SUBCLASSES(Class-set, Scope)` and `INTERSECTION_OF_SUPERCLASSES(Class-set, Scope)` are used to find the common subclasses and superclasses of classes (the argument `Class-set`) in a collection of basic classifications (the argument `Scope`). The operations `DIFFERENCE_OF_SUBCLASSES(Class-set1, Class-set2, Scope)` and `DIFFERENCE_OF_SUPERCLASSES(Class-set1, Class-set2, Scope)` are used to find those subclasses or superclasses of `Class-set1` which are not subclasses or superclasses of `Class-set2` within `Scope`.

For example, the following expression gives all classes which are subclasses of both classes `air-pollutant` and `carcinogene` :

```
INTERSECTION_OF_SUBCLASSES(
  {air-pollutant, carcinogene},
  {pollutant-types, pollutant-families, pollutants, hh-types,
   haz-compounds})
```

and yields the classes {`diesel-soot`, `coal-soot`, `asbestos`, `ash`}. Thus these operations give powerful support in the use of classes of multiple, poly-hierarchical classifications.

*C. Summary of the operations.* Our non-transitive operations can be summarized as follows :

*superclass-directed operations*

- `ROOT_CLASSES(Scope)` which yields the root classes within scope
- `IM_SUPERCLASSES(Class, Scope)` which yields the immediate superclasses of a class within scope

*subclass-directed operations*

- `LEAF_CLASSES(Scope)` which yields the leaf classes within scope
- `IM_SUBCLASSES(Class, Scope)` which yields the immediate subclasses of a class within scope

*non-directed operations*

- `CLASSES(Scope)` which yields all classes within scope
- `SET_INTERSECTION(set1, set2)` which yields the intersection
- `SET_UNION(set1, set2)` which yields the union, and
- `SET_DIFFERENCE(set1, set2)` which yields the difference.

Our transitive operations can be summarized as follows (all are directed) :

*superclass-directed operations*

- `SUPERCLASSES(Class, Scope)` gives all superclasses of a class within the scope
- `UNION_OF_SUPERCLASSES(Class-set, Scope)` computes all superclasses of a given set of classes within the scope
- `INTERSECTION_OF_SUPERCLASSES(Class-set, Scope)` computes common superclasses for a given set of classes within the scope
- `DIFFERENCE_OF_SUPERCLASSES(Class-set1, Class-set2, Scope)` which finds those superclasses of `Class-set1` which are not superclasses of `Class-set2` within the scope

*subclass-directed operations*

- `SUBCLASSES(Class, Scope)` gives all subclasses of a class within the scope
- `UNION_OF_SUBCLASSES(Class-set, Scope)` computes all subclasses of a given set of classes within the scope
- `INTERSECTION_OF_SUBCLASSES(Class-set, Scope)` computes common subclasses for a given set of classes within the scope and

- `DIFFERENCE_OF_SUBCLASSES(Class-set1, Class-set2, Scope)` which finds those subclasses of `Class-set1` which are not subclasses of `Class-set2` within the scope.

## 4.2. Nested deductive expressions

The deductive operations allow nested expressions where the result of one operation (or sub-expression) forms an argument of another operation. The BNF (Backus-Naur Form) grammar generating syntactically correct deductive expressions is given in Appendix II. In principle, an operation can be used to compute an argument of another operation provided that the computed argument is of the correct type : a single class when a single class is required and a class set when a set is required. For example, the following expression finds all subclasses of air-pollutant that are superclasses to both at least one greenhouse-gas and to at least one poison :

**Sample Query Q1** : Which air-pollutant classes may contain poisonous greenhouse-gases ?

```
SET_INTERSECTION(
  UNION_OF_SUPERCLASSES(
    SUBCLASSES(greenhouse-gas, {gh-gases}),
    {pollutant-types, pollutant-families, pollutants}),
  UNION_OF_SUPERCLASSES(
    SUBCLASSES(poison, {haz-compounds}),
    {pollutant-types, pollutant-families, pollutants}))
```

and yields {COx, CFC, gas-pollutant, air-pollutant}. In this expression, the innermost `SUBCLASSES` operations find all greenhouse-gases and all poisons, respectively. The first `UNION_OF_SUPERCLASSES` operation then computes the union all superclasses of greenhouse gases within the `air-pollutant` classification. The second `UNION_OF_SUPERCLASSES` operation computes the union of all superclasses of poisons within the `air-pollutant` classification. The outermost `SET_INTERSECTION` operation finally finds the intersection, i.e. classes which are superclasses of at least one greenhouse-gas and one poison. One implication of the result is that other more specific subclasses of air-pollutant do not contain poisonous greenhouse-gases while those reported may do so (in fact, in our sample case they do not).

The sample query Q1 above exemplifies two important features of queries in our operation-oriented query language for classifications : (1) although the operations themselves are *unidirectional* (either sub- or superclass-directed), the queries may be *unidirectional* or *bidirectional*, and (2) the queries may process one or more classifications in different directions. Queries can therefore be classified as *single* or *multiple classification queries* on one hand, and as *unidirectional* or *bidirectional queries* on the other. Thus our sample query Q1 is a bidirectional multiple-classification query. Below we give another sample query (Q2)

which is unidirectional in multiple classifications. Its result is illustrated in Fig. 4.1. Further examples and categorisation is given in Chapter 6 and, outside the classification context, in (Niemi & Järvelin, 1992c).

**Sample Query Q2** : Which are the metropolitan sensor-stations in Scotland ?

```
SET_INTERSECTION(
  SUBCLASSES(metropolitan, {gb-typed-stations}),
  SUBCLASSES(scotland, {gb-subregions, gb-locations, gb-station-
    loc}))
```

{edinburgh1, glasgow1, ...}

**Fig. 4.1.** The result of sample query Q2

Our operation-oriented query language for classifications provides end users with well-defined high-level operations. This essentially simplifies the use of transitive relationships. Rule-based Horn-clauses as the interface -- the standard interface in deductive databases -- for defining queries involving transitive computation would prevent end users from using deductive information retrieval. This is so because the user should formulate queries by specifying recursive rules. We have earlier considered factors which make recursive Horn-clauses too demanding for end users (Niemi & Järvelin, 1992c). In our approach, the recursive definitions needed in the implementation of the deductive operations are invisible to the user. In practise, high-level deductive operations are necessary for deductive IR in modern fact databases.

## 5. Integrated Query Language

Deductive classification-based operations alone, or conventional relational query languages alone have a relatively poor expressive power. It is their integration which provides a powerful query language for deductive information retrieval. The power stems from the possibility of retrieving data by some criteria, identifying the classes the data are related to, performing deduction on the basis of the classes in order to identify criteria for further data retrieval, etc. In other words, retrieval power stems from bidirectional multiple-classification deduction combined with intervening data retrieval. Section 5.1 summarizes relational algebra (RA). In Section 5.2 we present the integration approach of RA and our operation-oriented query language for classifications after which we consider the specifics of integration in Section 5.3.

## 5.1. Relational Algebra

As RA is well-known we give here only the syntax of the operations. Let  $Ex$ ,  $Ex1$  and  $Ex2$  be expressions consisting of either stored relation names or relational expressions constructed by the operations defined below. We consider the usual seven operations (see e.g. Maier, 1983; Ullman, 1988) :

- relational UNION( $Ex1$ ,  $Ex2$ ), INTERSECTION( $Ex1$ ,  $Ex2$ ), DIFFERENCE( $Ex1$ ,  $Ex2$ ), and PRODUCT( $Ex1$ ,  $Ex2$ ) ; note that the analogous ordinary set operations are called SET\_INTERSECTION, SET-UNION, etc.
- JOIN( $Ex1$ ,  $Ex2$ , Predicate), where Predicate is any Boolean predicate on the operand expression results
- RESTRICTION( $Ex$ , Predicate), where Predicate is any Boolean predicate on the operand expression result and
- PROJECTION( $Ex$ , AttributeNames), where AttributeNames is any subset of the attribute name set of the operand expression result.

The operations can be nested in more complex expressions, e.g. PROJECTION(JOIN(RESTRICTION(TEMPERATURE, TStat = exeter1), RESTRICTION(Pollutants, PStat = exeter1), TDay = PDay  $\wedge$  TMonth = PMonth  $\wedge$  TYear = PYear), {DAvTmp, Ptant, PMesment}). The BNF grammar generating syntactically correct RA expressions is given in Appendix II. The types of Boolean predicates considered in this paper can be learned from the grammar. It is worth noting that predicate primitives of the types :

- attribute-value-primitive, e.g. PStat = exeter1 and
  - one-of-primitive, e.g. Pollutant one-of({SO2, NO, NO2})
- are essential in the integration of the deductive and RA operations.

## 5.2. The Integration Approach

We integrate the two types of operations, deductive and RA operations, to form a general query language of great expressive power by (I) allowing the construction of sets of classes for class-oriented operations by relational expressions which yield a single-attribute result relation, (II) allowing the construction of classes for some class-oriented operations by integrated expressions which yield a single-attribute single-value result, (III) allowing deductive (or integrated) expressions in predicate primitives in order to construct values and value-sets for comparison.

The principles of integration are (a) extension of the BNF notations by new production rules, (b) definition and use of simple interface functions for simple type conversion when the mathematical type of the result given by a subexpression is different from the one assumed, and (c)

subexpression type checking during evaluation. We present the BNF extensions below and discuss the integration cases (I) - (III) in detail in Section 5.3.

The BNF grammar **<gen-expression>** for integrated query expressions is obtained by combining the grammars for **<rel-expression>** and **<class-expression>**, and by extending them by the following productions. The production **<gen-expression>** integrates the different expression types :

**<gen-expression>** → **<rel-expression>** | **<class-expression>**

This allows a general expression to be either relational or deductive.

*Integration Case I* : The production for **<class-operation>** below extends class-oriented deductive operations by allowing construction of class sets by relational expressions. This production alone does not enforce the requirement for a single-attribute result. Enforcing the requirement is discussed in Section 5.3.

**<class-operation>** → **<rel-expression>**

*Integration Case II* : The production for **<class>** below extends classes to general relational and deductive expressions. Thus the arguments of type **<class>** in operations `IM_SUBCLASSES`, `IM_SUPERCLASSES`, `SUBCLASSES`, `SUPERCLASSES` can be constructed by such expressions. This production alone does not enforce the requirement for a single-attribute and single-value result (see Section 5.3). The production for **<class-set>** disables class sets which are mixtures of atomic classes and general expressions. Allowing such mixtures would complicate query evaluation. It replaces the original production for **<class-set>** in Appendix II. The quoted braces in the BNF productions below are set delimiters.

**<class>** → **<atomic-class>** | **<rel-expression>** | **<class-operation>**  
**<class-set>** → `"{"<atomic-class>{,<atomic-class>}*"}"`

*Integration Case III* : The productions below make possible expressions where values to be compared in predicate primitives are constructed by general expressions. These productions replace the original productions **<numeric value>** and **<value>** in Appendix II. Note that a class can always be treated as an atomic value.

**<numeric value>** → **<number>** | **<gen-expression>**  
**<value>** → **<atomic value>** | **<gen-expression>**  
**<value-set>** → `"{"<value>{,<value>}*"}"` | **<class-operation>**

These BNF-extensions and modifications are also listed in Appendix II which thus contains the BNF grammar for the entire integrated query language. It allows very flexible mixing of relational and deductive operations at unlimited levels of nesting. Several examples of integrated expressions are given in Chapter 6.

### 5.3. Type Conversion Interfaces for the Integrated Query Language

*A. Integration Case I:* Relational expressions are useful for constructing sets of classes when the relational database is searched for specific classes occurring in the data. For example, one might want to find such pollutants whose air content has exceeded a certain critical limit at a given time and in a given geographic area. Integration of the expression types becomes necessary if some deduction should be performed on the identified classes. For example, one might want to know which pollutant families contain the pollutants identified in the relations. In such a case, a relational operation is used as a class-oriented operation in deduction.

As a specific case, consider the following RA expression which gives the pollutants (i.e. classes) whose air content has exceeded the critical content for forests at the station london1 in January 1991 (the database is in Fig. 2.2) :

```

PROJECTION(
  RESTRICTION(
    JOIN(
      RESTRICTION(POLLUTANTS,
        PSTAT = london1 ^ PYear = 1991 ^ PMonth= jan),
      RESTRICTION(CRITICALCONTENT,
        Recipient = forests),
      Ptant = Pollutant),
      PMesment > CritCont),
    {Pollutant})

```

The relation instance constructed by this expression is a relation of unary tuples, e.g. {<SO2>, <NO2>, ...} where the angle brackets denote the tuple delimiters. In order to find the pollutant families containing these pollutants the operation UNION\_OF\_SUPERCLASSES({SO2, NO2, ...}, {pollutants}) is needed. Integration requires that the RA expression can be used directly as an operand of the deductive operation. The result of the RA expression {<SO2>, <NO2>, ...} must therefore be transformed into a class set {SO2, NO2, ...}.

In general, whenever a relational expression is used as a class-oriented operation, the result relation of the expression must be converted into a set of classes. Therefore the result relation must contain only one attribute whose domain is a subset of **CLASS**. The type conversion function which checks this condition and transforms the single component tuples of the relation into a set of atomic values is easy to define (omitted here, see Järvelin & Niemi, 1992).

In spite of the general requirement on the attribute domain, expression compatibility can also be analyzed more precisely. Let  $Ex$  be the relational expression yielding a single-attribute result (in domain  $D$ ) which is used as a class-oriented argument in a class-oriented operation  $OPERATION(Ex, \{BC_1, \dots, BC_k\})$ . Let  $BC_i$  be a subset of the Cartesian product  $A_i \times B_i$  (i.e.  $A_i \times B_i \supseteq BC_i$ , **CLASS**  $\supseteq A_i \cup B_i$ ). If  $OPERATION$  is a superclass-directed operation (i.e.  $OPERATION \in \{UNION\_OF\_SUPERCLASSES, INTERSECTION\_OF\_SUPERCLASSES\}$ ), then there

must be some  $B_i$  such that  $B_i \supseteq D$ . If OPERATION is DIFFERENCE\_OF\_SUPERCLASSES(Ex1, Ex2, Scope) and D1 and D2 are the domains of the single attribute results of expressions Ex1 and Ex2, then there must be some  $B_i$  such that  $B_i \supseteq D1$  and some  $B_j$  such that  $B_j \supseteq D2$ . If OPERATION is a subclass-directed operation, the scope must contain some basic classification(s) such that D (or D1, D2) is a subset of its superclass domain. If these specific requirements are not met, i.e. if D (or D1, D2) does not match any of the subclass or superclass domains, the results of the operations are empty. However, because the subclass and superclass domains of the basic classifications are all subsets of **CLASS**, no run-time errors ensue if the general requirement is met. In our experimental system, only the general requirement is enforced.

*B. Integration Case II* : In this integration case a general expression, be it an RA or a class-expression, is used to construct a single value. Such a use of RA expressions may seem somewhat peculiar because RA expressions construct relations which are by their very nature sets of multi-component tuples. Such a use of class-expressions may also seem peculiar because class-expressions inherently produce sets of classes rather than single classes. However, this integration case is merely an extension of the *subquery* concept of SQL (e.g. Ullman, 1988). There are many expressions which can be guaranteed to produce a single-attribute single-tuple result.

For example, if one looks up the class name for DDT in the DICTIONARY-relation (projected on the Class attribute), there will be a single-attribute single-tuple result relation. The expression :

```
PROJECTION(
  RESTRICTION(DICTIONARY, PollName = DDT),
  {Class})
```

yields the result instance {<C14H9Cl5>} which can be used as a starting class for class-based computation. Further, the expression SUPERCLASSES(C14H9Cl5, {pollutant-types, pollutant-families, pollutants, hh-types, haz-compounds}) finds the more generic pollutant categories to which the identified pollutant C14H9Cl5 belongs (e.g. CxHxCly, gas-pollutant, air-pollutant, poison, health-hazard). This integration type is thus necessary.

Allowing the construction of single classes for class-oriented operations by class- and RA expressions requires, in principle, that these expressions produce a single value in some way, e.g. as a single element set or as a single-attribute - single-tuple relation. In such cases the user's intended query semantics is quite obvious and type conversion is simple. However, it is difficult to determine in all cases in advance, on the basis of a general expression, whether the result fulfils this requirement. Three approaches to the problem seem possible. Either no methods for handling cases violating the requirement are defined (with the consequence of

run-time errors in violating cases), or the system is made more robust by some methods for handling type violations, or then the user is provided with operators (e.g. *min*, *max*, *some*) for selecting one value among the possibly many given by a subexpression.

Some automatic methods for handling type violations seem quite reasonable, e.g. turning the expressions `SUBCLASSES(Expr, Scope)` to expressions `UNION_OF_SUBCLASSES(Expr, Scope)` if `Expr` returns a multi-element set instead of a single value. However, it is not always clear how similar type errors (e.g. in the operation `IM_SUBCLASSES`) should be handled. Further study is thus required. Serious user errors should not be hidden by the system's assumptions. In the following we assume that the subexpressions return single-values as discussed above. Thus only the following simple type conversion functions *rel-to-one-value* and *set-to-one-value* are needed for converting a relation and a set, respectively, into one value. They require that the arguments are single-values as discussed above. Otherwise they are undefined.

*Definition 5.1* : Let  $\{v\}$  be any set consisting of one atomic element  $v$ . The function *set-to-one-value* is defined as  $set-to-one-value(\{v\}) = v$ . Let  $\{<v>\}$  be any single-attribute - single-tuple relation instance. The function *rel-to-one-value* is defined  $rel-to-one-value(\{<v>\}) = v$ .

When an argument class of a class-oriented operation is produced by a general expression, the type of the expression result (relation or set) can be recognized from its topmost operation. Thus the appropriate type conversion function can be determined automatically in query evaluation. This is done as illustrated below by the function *value-conversion* which combines the type conversion functions so that any integrated expressions are evaluated and converted to a single value. If the argument expression is already a single value, it is returned. The function *atomic* discerns atoms from tuples, sets and other compound structures by yielding "true" for the former and "false" for the latter. The subfunctions *rel-evaluate* and *class-evaluate* are the evaluators for the respective expression types recognized by the type-checking functions *rel-expression* and *class-operation* (see below).

*Definition 5.2* : Let  $G$  be any constant or a general expression,  $rdb$  a relational database, and  $CB$  the classification base. The function *value-conversion* is :

$$value-conversion(G, rdb, CB) = \begin{cases} G & \text{if } atomic(G), \\ rel-to-one-value(rel-evaluate(G, rdb, CB)) & \text{if } rel-expression(G), \\ set-to-one-value(class-evaluate(G, rdb, CB)) & \text{if } class-operation(G). \end{cases}$$

For example,  $value\text{-}conversion(5, RDB1, CB1) = 5$ , while  $value\text{-}conversion(PROJECTION(RESTRICTION(DICTIONARY, PollName = DDT), \{Class\}), RDB1, EB1) = C14H9C15$ . The expression  $value\text{-}conversion(PROJECTION(RESTRICTION(DICTIONARY, PollName = DDT), \{PollName, Class\}), RDB1, EB1)$  is undefined because the RA expression yields a relation containing two attributes.

*C. Integration Case III.* In SQL, it is possible to compute value sets and single values for predicate primitives by subqueries, which are any SQL expressions yielding a single-attribute result (or a single-attribute single-value result in some cases, e.g. Ullman, 1988). Moreover, SQL provides the operators ANY and ALL in addition to IN and NOT IN which correspond to our primitive types (AttrName one-of(Set)) and  $\neg$  (AttrName one-of(Set)). Here we extend the subquery concept from RA to general expressions covering both RA and class-expressions.

Allowing general expressions in predicate primitives to construct the values for comparison requires, in principle, that the expression produces in some way a single value, as discussed above. In this type of integration, it is also difficult to determine in all cases and in advance, on the basis of an integrated expression, whether its result fulfils this requirement.

In this integration case the system could also be made more robust in handling type inconsistencies. For example, the syntactically incorrect primitive  $aname < \{v1, \dots, vn\}$  could be normalized by selecting the minimum among  $v1, \dots, vn$ , whereas the syntactically incorrect primitive  $aname = \{v1, \dots, vn\}$  could be normalized by replacing it by  $aname \text{ one-of}(\{v1, \dots, vn\})$ , etc. While some of these possibilities seem quite interesting and easy to implement, further study on the desirability of such robustness is required. We assume below that the subexpressions return single-values as discussed above.

When a value component of a predicate primitive is produced by a general expression, the type of the expression result can be recognized from its topmost operation. The appropriate type conversion function can then be determined by the function *value-conversion*.

We shall now define a Boolean predicate normalization method for predicates containing general expressions in components where atomic values are normally expected. This method is implemented by the function *normalize-primit* which normalizes all primitives of Boolean predicates with the help of the function *value-conversion* (Def. 5.2). Consider a primitive  $aname < G$ , where  $G$  is a general expression. In the normalization,  $G$  is evaluated and converted by *value-conversion* to the normalized primitive  $aname < val$ , where  $val = value\text{-}conversion(G, rdb, CB)$ . If  $G$  does not yield a single-value, the transformation is undefined. In the definition, the function *is-set*(Set) tests whether the argument Set is a set according to the grammar **<value-set>** of the integrated grammar (Appendix II).

*Definition 5.3* : Let  $p$  be any predicate primitive of the integrated grammar (i.e. its value components may be constants or general expressions),  $rdb$  the relational database, and  $CB$  the classification base. The function *normalize-primit* is :

$$\begin{aligned}
 & \text{normalize-primit}(p, rdb, CB) = \\
 & \quad (\text{aname } \theta \text{ value-conversion}(G, rdb, CB)) \\
 & \quad \quad \text{if } p = (\text{aname } \theta \text{ } G), \\
 & \quad (\text{aname between}(\text{value-conversion}(G1, rdb, CB), \text{value-conversion}(G2, rdb, \\
 & \quad \quad \quad \text{CB}))) \\
 & \quad \quad \text{if } p = (\text{aname between}(G1, G2)), \\
 & \quad (\text{aname one-of}(\{\text{value-conversion}(g, rdb, CB) \mid g \in \text{Set}\})) \\
 & \quad \quad \text{if } p = (\text{aname one-of}(\text{Set}) \wedge \text{is-set}(\text{Set}), \\
 & \quad (\text{aname one-of}(\text{class-evaluate}(\text{Ex}, rdb, CB))) \\
 & \quad \quad \text{if } p = (\text{aname one-of}(\text{Ex}) \wedge \text{class-operation}(\text{Ex}), \\
 p & \quad \text{if } p = \text{true} \vee p = \text{false} \vee (p = (\text{aname1 } \theta \text{ aname2}) \wedge \text{atomic}(\text{aname1}) \wedge \\
 & \quad \quad \text{atomic}(\text{aname2}))
 \end{aligned}$$

In the first case, the predicate primitive  $p = (\text{aname } \theta \text{ } G)$  requires that the attribute name  $\text{aname}$  is in the relation  $\theta$  ( $\theta \in \{<, \bullet, =, >, \circ, \cdot\}$ ) to  $G$ . The resulting predicate is  $(\text{aname } \theta \text{ } G')$  where  $G'$  is derived from  $G$  by the function *value-conversion*( $G, rdb, CB$ ). The second case is otherwise similar but converts both limits of the range for a 'between' -type of predicate. In cases three and four, predicates of type 'one-of' are converted. In case three the argument  $\text{Set}$  of  $\text{one-of}(\text{Set})$  is confirmed as a set and then all of its elements are converted one by one. In case four it is required that the argument  $\text{Ex}$  of  $\text{one-of}(\text{Ex})$  is a class-operation which is then evaluated directly by the function *class-evaluate*. In the last case, primitives which need no normalization are left intact.

For example, let  $G1$  denote the expression  $\text{PROJECTION}(\text{RESTRICTION}(\text{TEMPERATURE}, (\text{TStat} = \text{london1}) \wedge (\text{TDay} = 12) \wedge (\text{TMonth} = \text{jan}) \wedge (\text{TYear} = 91)), \{\text{DAvTemp}\})$  which gives the average daily temperature of the station  $\text{london1}$  on January 12th, 1991 (cf. our sample RDB in Fig. 2.2). Now  $\text{normalize-primit}((\text{TMaxTemp} < G1), \text{RDB1}, \text{CB1}) = (\text{TMaxTemp} < 33)$ . As the second example, assume that data on poisonous CFCs and chlorinated hydrocarbons are to be retrieved. The compounds are identified by the expression  $G2 = \text{SET\_INTERSECTION}(\text{UNION\_OF\_SUBCLASSES}(\{\text{CFC}, \text{HxCxC1x}\}, \{\text{pollutant-families}, \text{pollutants}\}), \text{LEAF\_CLASSES}(\{\text{haz-compounds}\}))$  which yields  $\{\text{C3HC14F}, \text{C2HC15}, \text{C14H9C15}\}$ . The data retrieval primitive is constructed by  $\text{normalize-primit}((\text{Ptant one-of}(G2), \text{RDB1}, \text{CB1}) = (\text{Ptant one-of}(\{\text{C3HC14F}, \text{C2HC15}, \text{C14H9C15}\}))$ .

The function *normalize-pred* normalizes general Boolean predicates by normalizing all their primitives without changing the predicate structure.



relational and deductive expressions is also considered in (Niemi & Järvelin, 1992b). The full integration of three classes of operations was defined in (Järvelin & Niemi, 1992). These include extended RA, deductive and entity-based (Järvelin & Niemi, 1993) operations. Necessary knowledge representations and implementation aspects are presented in these publications in detail. The evaluators sketched in Appendix III are reduced in scope for the classes of operations considered in this paper, i.e. ordinary RA and class-operations. All details of definition cannot be presented here -- the interested reader is referred to the publications mentioned above. Below we shall consider briefly the main ideas behind the evaluators.

Every operation of the integrated query language has its own evaluation function. For example, the evaluation function for the restriction operation is the function *restriction*(Rel, Pred) with Rel, a relation, and Pred, a predicate, as its arguments (Niemi & Järvelin, 1985). This means that whenever during evaluation the expression `RESTRICTION(Ex1, P)` is recognized as the top-most operation of a query, Ex1 and P are first evaluated (if not already in the fully evaluated form) into Rel and Pred and the function *restriction*(Rel, Pred) is then applied to the result. The case for restrictions in the function *rel-evaluate*(Ex, rdb, CB) is therefore (see Appendix III):

$$\begin{aligned} & \textit{restriction}(\textit{rel-evaluate}(\text{Ex1}, \text{rdb}, \text{CB}), \textit{normalize-pred}(\text{P}, \text{rdb}, \text{CB})), \\ & \text{if } \text{EX} = \text{RESTRICTION}(\text{Ex1}, \text{P}) \end{aligned}$$

The evaluation of the relational argument Ex1 requires recursive application of the function *rel-evaluate* and the evaluation of the predicate argument P requires application of the normalization function *normalize-pred*.

Similarly, the evaluation function for the subclasses -operation is the function *subclasses*(Class, ScopeBCs) with Class, a class name, and ScopeBCs, the scope of basic classifications considered, as its arguments (defined under the name *successors* in Niemi & Järvelin, 1992a). Whenever the expression `SUBCLASSES(Ex1, Scope)` is encountered during evaluation as the topmost operation, Ex1 is first evaluated (if not already in the fully evaluated form) into Class, the basic classifications referred by Scope are retrieved into ScopeBCs, and then the function *subclasses*(Class, ScopeBCs) is applied to the result. Therefore the case for the subclasses operation in the function *class-evaluate* is (see Appendix III) :

$$\begin{aligned} & \textit{subclasses}(\textit{value-conversion}(\text{Ex1}, \text{rdb}, \text{CB}), \textit{scope-BCs}(\text{Scope}, \text{CB})), \\ & \text{if } \text{EX} = \text{SUBCLASSES}(\text{Ex1}, \text{Scope}) \end{aligned}$$

The evaluation of the class-argument Ex1 requires transformation by the function *value-conversion* (Ex1 may be a relational expression !). Note that the evaluator *class-operation* is

also used recursively, either indirectly via the function *value-conversion*, or directly as in the transitive deductive operations (e.g. `UNION_OF_SUBCLASSES`). The function *scope-BCs*(Scope, CB) returns the basic classifications given in Scope from CB.

The evaluation of other operations is similar. The functions *rel-evaluate* and *class-evaluate* evaluate complex expressions recursively and call each other indirectly via other functions (e.g. *normalize-pred*) or directly whenever a need for integrating different classes of operations arises. In the function *rel-evaluate*, integration is accomplished through predicates in the operations join and restriction. In the function *class-evaluate*, integration is achieved either by direct conversion of a relation (in case 2) or in many other cases through the function *value-conversion*.

## 6. Queries in the Integrated Language

In this chapter we demonstrate how complex retrieval tasks can be carried out by integrated expressions. We organize the sample queries according to different types of deductive queries : *unidirectional* or *bidirectional queries* and *single-* or *multiple-classification queries*. We shall indicate the type of integration used in each case. Note that the sample query results are only illustrative -- they are compatible with the sample database RDB1 which, however, is too small to provide all the data assumed in the sample evaluations.

### 6.1. Bidirectional Single-Classification Queries

Assume that a user wants to retrieve the mean temperature for Jan 15, 1991 for all sensor stations in the region to which Sheffield belongs. This query involves finding first the region for Sheffield (in the superclass direction) and then the sensor stations in the region (in the subclass direction). Thereafter the set of sensor stations can be used to retrieve the data for the stations and date. The link between deductive expressions and RA operations in this case is a predicate primitive of type (AttrName one-of(Set)), i.e. this query belongs to Integration Case III. In the sample query **Q3**, the region of Sheffield is retrieved by `SUPERCLASSES(sheffield, {gb-locations, gb-subregions})` which yields {mid-eng-land, england}. The sensor stations in the area are retrieved by `UNION_OF_SUBCLASSES({mid-england, england}, {gb-subregions, gb-locations, gb-stat-names})` which yields a set of sensor station names in many cities, including Sheffield, Birmingham, and Stoke. The TEMPERATURE data for these stations and January 15, 1991 are selected and projected. The result is outlined in Fig. 6.1.

**Sample Query Q3 :** Mean temperature of Jan 15, 1991 at all sensor stations in the Sheffield region

```

PROJECTION(
  RESTRICTION(TEMPERATURE,
    TStat one-of(
      UNION_OF_SUBCLASSES(
        SUPERCLASSES(Sheffield, {gb-locations,
          gb-subregions}),
          {gb-subregions, gb-locations,gb-stat-names}))
    ^ TDay = 15 ^ TYear = 1991 ^ TMonth = jan),
  {DAvTmp, TStat}).

```

DAvTmp	TStat
35	sheffield1
32	sheffield2
40	birmingham2
43	stoke5
...	...

**Fig. 6.1.** Sample result relation for query Q3

Let us next assume that a user wants to retrieve measurements of the air-contents of any Chlor-Fluor-Carbons in the London area. All CFCs are identified by the expression `SUBCLASSES( PROJECTION(RESTRICTION(DICTIONARY, PollName = Chlor-Fluor-Carbons), {Class}), {pollutant-families, pollutants})`. The RA expression in the expression returns a single-attribute single-tuple result and thus this expression belongs to Integration Case II. The whole expression yields the set of CFCs as {C2H5F, C3HC14F, C2C14F2, C2C12F4}. The London sensor-stations are retrieved by `SUBCLASSES(london, {gb-stat-names})`. These deductive expressions are used in the restriction predicate (Integration Case III). The restriction and projection operations find the final result illustrated in Fig. 6.2.

**Sample Query Q4 :** Measurements of air-contents of any CFCs in the London area

```

PROJECTION(
  RESTRICTION(POLLUTANTS,
    PStat one-of(SUBCLASSES(london, {gb-stat-names})) ^
    Ptant one-of(
      SUBCLASSES(
        PROJECTION(
          RESTRICTION(DICTIONARY,
            PollName = Chlor-Fluor-Carbons),
            {Class}),
          {pollutant-families, pollutants}))),
    {Ptant, PMesment, PStat})

```

Ptant	PMesment	PStat
C2H5F	0.0000004	london1
C2H5F	0.0000002	london2
C3HCl4F	0.0000010	london1
C3HCl4F	0.0000012	london2
...	...	...

**Fig. 6.2.** Sample result relation for query Q4

## 6.2. Unidirectional Multiple-Classification Queries

Next we consider a sample case where two classifications are used to narrow down the data to be retrieved. Assume that the temperature and chlorinated hydrocarbon (CHC) readings are needed for all sensor stations reporting any chlorinated hydrocarbons in East England in winter 1991. The CHCs are identified by the expression `SUBCLASSES(HxCxC1x, {pollutants})` which yields the set `{C2H5C1, C2HC15, C14H9C15}`. The sensor stations in East England are retrieved by `SUBCLASSES(east-england, {gb-locations, gb-stat-names})` which yields a set of sensor station names for several cities, including Harwich and Cambridge. The expression is below and its result relation is illustrated in Fig. 6.3. Obviously, all integration in this expression belongs to case III.

**Sample Query Q5 :** Temperature and CHC readings for sensor stations reporting any CHCs in East England in winter 1991

```

PROJECTION(
  JOIN(
    RESTRICTION(TEMPERATURE,
      TStat one-of(
        SUBCLASSES(east-england,
          {gb-locations, gb-stat-names}))
        ^ TYear = 1991 ^ TMonth one-of({jan, feb})),
    RESTRICTION(POLLUTANTS,
      Ptant one-of(
        SUBCLASSES(HxCxC1x, {pollutants})) ^
        PStat one-of(
          SUBCLASSES(east-england,
            {gb-locations, gb-stat-names})) ^
          PYear = 1991 ^ PMonth one-of({jan, feb})),
      TStat = PStat ^ TDay = PDay ^ TMonth = PMonth ^
      TYear = PYear),
    {DAvTmp, PMesment, Ptant, TStat, TDay, TMonth}).

```

DAvTmp	PMesment	Ptant	TStat	TDay	TMonth
35	0.0000001	C2H5Cl	harwich1	12	jan
32	0.0000002	C2H5Cl	harwich2	12	jan
40	0.0000001	C2H5Cl	cambridge1	15	jan
43	0.0000003	C2HCl5	harwich1	15	feb
...	...	...	...	...	...

**Fig. 6.3.** Sample result relation for query Q5

### 6.3. Bidirectional Multiple-Classification Queries

Bidirectional multiple-classification queries make it possible to proceed to the subclass direction in one classification and to the superclass direction in another. Assume that the user wants to know what types of serious health hazards have been caused by gas-pollutants in the London area recently. A health hazard caused by a pollutant is considered serious if its air content has recently exceeded the critical limit for humans at least once. "Recently" is taken to mean the year 1991. The pollutants belonging to the family gas-pollutants are identified by the expression `SUBCLASSES(gas-pollutant, {pollutant-families, pollutants})` which yields the set `{COx, CO, CO2, NOx, NO, NO2, NO3, Ozone, O3, SOx, SO, ...}`. The sensor stations in the London area are given by the expression `SUBCLASSES(london, {gb-stat-names})`. The `POLLUTANTS` relation is searched for the London gas-pollutant readings in 1991 and the relation `CRITICALCONTENTS` for the critical contents of gas-pollutants for humans. The results are joined by pollutant and possible excess of critical limits is checked. Tuples passing the test are projected. Finally, going in the superclass-direction, the health-hazard types are identified by the expression `UNION_OF_SUPERCLASSES(PollSet, {haz-compounds})`, where `PollSet` is the set of pollutants whose air content exceeded the critical limit. The query expression Q6 is below and its result set in Fig. 6.4. In this expression, the restriction predicates are examples of Integration Case III while the top-most operation is an example of Integration Case I.

**Sample Query Q6 :** Which types of health hazards have been caused by gas-pollutants in the London area recently ?

```

UNION_OF_SUPERCLASSES(
  PROJECTION(
    JOIN(
      RESTRICTION(POLLUTANTS,
        PStat one-of(
          SUBCLASSES(london, {gb-stat-names})) ^
          Ptant one-of(
            SUBCLASSES(gas-pollutant,
              {pollutant-families, pollutants})) ^
          TYear = 1991),
      RESTRICTION(CRITICALCONTENTS,
        Pollutant one-of(

```

```

SUBCLASSES(gas-pollutant,
            {pollutant-families, pollutants})) ^
Recipient = human),
Ptant = Pollutant ^ PMesment > CritCont),
{Ptant}),
{haz-compounds})

```

{allergene, poison}

**Fig. 6.4.** Sample result set for Query **Q6**

A very brief but informative answer about recent London health hazards is obtained through a relatively simple and powerful expression. The query traverses the geographic classification toward the subclasses to find the London sensor stations and the air-pollutant classification towards the subclasses to identify all gas-pollutants. It then retrieves the data associated with these classes in the relation POLLUTANTS. It likewise retrieves the critical air-content levels for all gas-pollutants. Next it checks the data for values exceeding the critical levels. Finally it proceeds towards the superclasses in the health-hazard classification to identify the health-hazard types associated with high air-content pollutants. Deduction and fact retrieval alternate during query evaluation.

#### 6.4. Complex Queries

In our query language, any number of classifications may be consulted in either direction and in as many phases as necessary. Consider a complex case where the user wants to know which urban and metropolitan areas have recently been troubled by health hazard-types similar to those in the London area. The sample query **Q6** above gives the London area health hazards as {allergene, poison}. To identify any comparable risk cities, all pollutants causing these health hazards must be identified. These are given by the expression UNION\_OF\_SUBCLASSES(**Q6**, {haz-compounds}). Note that these pollutants need not be the same as in London. The relations POLLUTANTS and CRITICALCONTENTS are restricted and joined to form a relation containing tuples for sensor stations reporting recent pollution readings above the critical limits in the same way as in sample query **Q6**. The sensor stations are then projected and intersected with urban and metropolitan sensor stations. Finally the cities containing these stations are identified in the superclass-direction. The query expression **Q7** is below and its result set is illustrated in Fig. 6.5. Integration here belongs to cases I and III.

**Sample Query Q7 :** Which urban and metropolitan areas have recently been troubled by health hazard-types similar to those in the London area ?

```

UNION_OF_SUPERCLASSES(

```

```

SET_INTERSECTION(
  PROJECTION(
    JOIN(
      RESTRICTION(POLLUTANTS,
        Ptant one-of(UNION_OF_SUBCLASSES(
          Q6, {haz-compounds})) ^
          TYear = 1991),
      RESTRICTION(CRITICALCONTENTS,
        Recipient = human),
        Ptant = Pollutant ^ PMesment > CritCont),
      {PStat}),
    UNION_OF_SUBCLASSES({urban, metropolitan}, {gb-typed-
      stations}),
    {gb-stat-names})

```

{glasgow, manchester, liverpool, london, ...}

**Fig. 6.5.** Sample result set for Query Q7

After evaluating subquery Q6, the main part of Q7 goes towards the subclasses in the health-hazard classification to identify all pollutants causing similar effects and finds the pollution data for all of them and for all sensor stations in 1991. After checking critical limits and projecting on sensor stations, the query identifies all urban or metropolitan stations among them and then goes toward the superclasses in the geographic classification to identify relevant cities. Four classifications (pollutants, health-hazards, station-types and station-locations) are used in several phases and two of them bidirectionally. Again, deduction and fact retrieval alternate seamlessly during query evaluation.

## 7. Discussion

The deductive information retrieval language presented in Chapter 5 integrates existing multiple classifications available in fact databases effectively and seamlessly with fact retrieval. It allows unlimited levels of nesting of deductive and conventional operations. In other words, deductive and fact retrieval phases alternate seamlessly during evaluation. The language also integrates methods of deductive data management, fact retrieval and traditional IR in one powerful framework and shows how existing classifications can be utilized in a new, deductive way. This was demonstrated by the sample queries which process single or multiple classifications uni- or bidirectionally in several phases connected by intervening conventional retrieval.

The deductive information retrieval language presented is at a *very high abstraction level* because all recursive definition needed in transitive computation is hidden from the users (Niemi & Järvelin, 1992c). Although only the use of relational algebra was considered here, it is also simple to combine the deductive operations with SQL. The language provides *seamless inte-*

*gration* because the interfaces between conventional and deductive operations are well-defined. It *supports users' intuition* in the manipulation of the classifications because the representation of classifications and the deductive operations are defined in terms of classification terminology. The language essentially simplifies query formulation and provides an expressive power which so far has been available only for users with advanced programming skills. The language also provides the users with a homogeneous operation-oriented way of query formulation, clear semantics, and safe and modular query formulation (Niemi & Järvelin, 1992c).

Our approach extends traditional IR methods which provide only one unidirectional transitive operation on classifications, the hierarchical term expansion in one classification -- it is a special case among our deductive operations. Traditional IR methods lack among others operations for inferencing in the superclass direction. Although only classifications were considered in this paper, our methods are also valid in a thesaurus-based environment (see the sample relation DICTIONARY) to enhance terminological manipulation as proposed by Hoppe & al. (1990). Classifications available in databases form an environment for deductive techniques which has not as yet been sufficiently investigated.

We defined the representation of multiple hierarchical classifications as collections of connected basic classifications. This representation facilitates deduction and has several advantages : the user can limit deduction precisely to relevant classifications and hierarchical levels, and poly-hierarchical class relationships can be utilized. These features are necessary for deductive information retrieval but are not provided by current traditional IR systems or deductive databases.

The approach and the methods here proposed for deductive information retrieval have a wide area of application. They add deductive intelligence to bibliographic retrieval exceeding the capabilities of term frequency counts (e.g. the ZOOM operation, ESA, n.d.). The application of these methods in bibliographic retrieval entails the development of bibliographic data classifications which transcend mere document content (e.g. authors, their affiliations, and publishers). Text retrieval can also be enhanced when documents contain classified information. For example, hospital patient record texts may contain specific work-based illness categories, job categories, etc. which are based on authoritative classifications. A user may want to know which illness categories are reported together with certain kinds of jobs in the records. In many cases the relational model is promising for text retrieval (MacLeod, 1991). The enhancements to the relational model considered in the present article further improve its usefulness in text retrieval.

The approach presented in this article requires that users understand the relationships of basic classifications in the classification base and also the semantics of the deductive operations.

This is the problem with increased expressive power in automatic deduction and control over deduction. The former can be alleviated by well-designed interfaces but the latter is a skill to be learned. However, an SQL-like declarative version of expressing deduction is worth studying. Another current limitation from the classification processing viewpoint is that the transitive operations yield super- or subclasses of given starting classes at the target level of deduction *as well as* at all intermediate levels -- e.g. in sample query **Q3** the superclasses of Sheffield are {mid-england, england} while only the latter is needed for consequent subclass-directed deduction. Although the intermediate level classes can be removed by more complex expressions, the operations should have an option to drop them automatically during deduction.

We have shown earlier how data aggregation and data value conversion operations can be integrated with RA and deductive operations and have considered the increased expressive power of such a query language (Järvelin & Niemi, 1991 ; 1992). The integrated language and its evaluator have been implemented in Prolog in a workstation environment. The prototype implementation is described in detail in Järvelin & Niemi (1992) and Niemi & Järvelin (1991; 1992a-b).

## 8. Conclusions

In this article, we have shown how useful and widely applicable deductive information retrieval is, and have presented a powerful deductive information retrieval language which integrates fact retrieval, existing classifications and deductive methods seamlessly and at a high-level. We have defined the representation of classification information as collections of binary relations which allow the computation of transitive class relationships. The representation is suited to multiple multi-level hierarchical classifications. The deductive operations for classifications were introduced in classification-specific terms. Application specific terminology enhances user's intuition in the use of the operations. Possibilities for applying the deductive information retrieval language outside fact retrieval were indicated.

The aim of our approach to IR is to provide operations and knowledge representations, which reduce end-user effort in query formulation and increase end-user capability in deductive information retrieval. The deductive information retrieval language presented in this paper supports user adaptation to database environments containing classified information and further enhances data access. The language combines traditional IR and deductive methods in fact retrieval and shows that traditional IR and fact retrieval share a host of common user and research problems.

## References

- Bancilhon, F. & Ramakrishnan, R.(1988). Performance Evaluation of Data Intensive Logic Programs. In J. Minker (Ed.), *Deductive Databases and Logic Programming* (pp. 439-517). Los Altos, CA: Morgan Kaufman.
- ESA, Information Retrieval Service (*sine anno*). *QUEST User manual : The Searcher's Guide to Online Commands*. Frascati, Italy : ESRIN.
- Chen, C. & Herson, P. (Eds.). (1984). *Numeric Databases*. Norwood, NJ: Ablex.
- Hoppe, K. & Ammersbach, K. & Lutes-Schaab, B. & Zinssmeister, G. (1990). EXPRESS : An Experimental Interface for Factual Information Retrieval. In J-L. Vidick (Ed.), *Proc. of the 13th International Conference on Research and Development in Information Retrieval (ACM SIGIR '91)* (pp. 63-81). Brussels, Sept. 5-7, 1990. Bruxelles: ACM.
- Harter, S.R. (1986). *Online information retrieval : concepts, principles, and techniques*. Orlando, FL: Academic Press.
- Jagadish, H.V. & Agrawal, R. (1987). A Study of Transitive Closure as a Recursion Mechanism. In U. Dayal & I. Traiger (Eds.), *Proceedings of 1987 ACM SIGMOD International Conference on the Management of Data* (pp. 331-334). New York: ACM Press.
- Järvelin, K. (1988). A Methodology for User Charge Estimation in Numeric Online Databanks. Part I : A Review of Numeric Databanks and Charging Principles. *Journal of Information Science*, 14, 3-16.
- Järvelin, K. & Niemi, T. (1990). Simplifying Retrieval from Distributed Heterogeneous Fact Databases. Part I : Problems and Requirements. *International Forum for Information and Documentation*, 15, 8-15.
- Järvelin, K. & Niemi, T. (1991). Data Conversion, Aggregation and Deduction for Advanced Retrieval from Heterogeneous Fact Databases. In Bookstein, A. et al (Eds.), *The 14th International Conference on Research and Development in Information Retrieval (ACM SIGIR '91)*, Chicago, IL, Oct. 13-16, 1991, (pp. 173-182). New York, NY: ACM.
- Järvelin, K. & Niemi, T. (1992). *General Data Conversion and Aggregation Operations : Definition and Integration with Relational, Entity-Based and Deductive Data Retrieval Techniques*. Tampere, Finland: University of Tampere, Department of Computer Science, Report Series A-1992-3.
- Järvelin, K. & Niemi, T. (1993). An Entity-Based Approach to Query Processing in Relational Databases. Part I : Entity Type Representation. *Data & Knowledge Engineering*, 1993, in press.
- Lancaster, F.W. (1986). *Vocabulary Control for Information Retrieval*. 2nd ed. Arlington, VA: Information Resources Press.
- Maier, D. (1983). *The Theory of Relational Databases*. Rockville, MD: Computer Science Press.

- McLean, S. & Weise, C. (1991). Digress : A Deductive Interface to a Relational Database. *Journal of the American Society for Information Science*, 42, 49-63.
- MacLeod, I. (1991). Text Retrieval and the Relational Model. *Journal of the American Society for Information Science*, 42, 155-165.
- Niemi, T. & Järvelin, K. (1985). A Straightforward Formalization of the Relational Model. *Information Systems*, 10(1), 65-76.
- Niemi, T. & Järvelin, K. (1991) Prolog-based Meta-rules for Relational Database Representation and Manipulation. *IEEE Transactions on Software Engineering*, 17, 762-788.
- Niemi, T. & Järvelin, K. (1992a) Operation-Oriented Query Language Approach for Recursive Queries – Part 1. Functional Definition. *Information Systems*, 17(1), 49-75
- Niemi, T. & Järvelin, K. (1992b) Operation-Oriented Query Language Approach for Recursive Queries – Part 2. Prototype Implementation and Its Integration with Relational Databases. *Information Systems*, 17(1), 77-106.
- Niemi, T. & Järvelin, K. (1992c). Advanced Query Formulation in Deductive Databases. *Information Processing & Management*, 27, 181-199.
- Parsaye, K. & Chignell, M. & Khoshafian, S. & Wong, H. (1989). *Intelligent Databases : Object-Oriented, Deductive Hypermedia Technologies*. New York, NY: Wiley.
- Peat, H.J. & Willett, P. (1991). The Limitations of the Term Co-Occurrence Data for Query Expansion in Document Retrieval Systems. *Journal of the American Society for Information Science*, 42, 378-383.
- Tenopir, C. & Ro, J.S. (1990). *Full Text Databases*. New York, NY: Greenwood Press. (New Directions in Information Management : 21).
- Ullman, J.D. (1988). *Principles of Database and Knowledge Base Systems. Vol. I*. Rockville, MD: Computer Science Press.
- Ullman, J.D. (1989). *Principles of Database and Knowledge Base Systems. Vol. II : The new technologies*. Rockville, MD: Computer Science Press.
- Wolski, K. & Koivula, T. & Lukkari, M. & Mäkinen, M. (1991). *Data on Display : A Guide to Numeric Databases*. Helsinki, Finland: Central Statistical Office of Finland, Handbooks 25 B.
- Zloof, M.M. (1975). *Query-By-Example: Operations on Transitive Closure*. Yorktown Heights, NY: IBM, RC 5526.

## Appendix I : Sample Classifications

<b>air-pollutant</b>		NOx: NO, NO2, NO3
solid-pollutant		Ozone: O3
soot:	diesel-soot, coal-soot	SOx: SO, SO2, SO3
dust:	asbestos-dust, ash-dust	CFC: C2H5F, C3HCl4F, C2Cl4F2, C2Cl2F4
gas-pollutant		HxCxClx:
COx:	CO, CO2	C2H5Cl, C2HCl5, C14H9Cl5

Domains :

- root domain = air-pollutant
- second level domain = poll-type
- third level domain = poll-family
- leaf domain = pollutant

Binary Relations :

- pollutant-types : air-pollutant × poll-type
- pollutant-families : poll-type × poll-family
- pollutants : poll-family × pollutant

**health-hazard**

allergene: NO<sub>2</sub>, SO<sub>2</sub>, SO  
 carcinogene: diesel-soot, coal-soot, asbestos-dust, ash-dust  
 poison: CO, O<sub>3</sub>, NO, NO<sub>3</sub>, C<sub>2</sub>HCl<sub>5</sub>, C<sub>3</sub>HCl<sub>4</sub>F, C<sub>14</sub>H<sub>9</sub>Cl<sub>5</sub>

Domains :

- root domain = health-hazard
- second level domain = hh-type
- leaf domain = pollutant

Binary Relations :

- hh-types : health-hazard × hh-type
- haz-compounds : hh-type × pollutant

**greenhouse-gas**

CO<sub>2</sub>  
 C<sub>2</sub>Cl<sub>4</sub>F<sub>2</sub>  
 C<sub>2</sub>Cl<sub>2</sub>F<sub>4</sub>

Domains :

- root domain = greenhouse-gas
- leaf domain = pollutant

Binary Relation :

- gh-gases : greenhouse-gas × pollutant

**gb-sensor-type**

metropolitan: london1, london2, ..., birmingham1, sheffield1, edinburg1, glasgow1, ...  
 urban: london5, sheffield3, exeter1, londonderry1, ...  
 countryside: cambridge1, aberystwyth1, ...

Domains :

- root domain = sensor-type
- second level domain = station-type
- leaf domain = gb-stat-name

Binary Relations :

- station-types : sensor-type × station-type
- gb-typed-stations : station-type × gb-stat-name

**gb-station-loc**

london: london1, london2, ...  
 birmingham: birmingham1, ...  
 sheffield: sheffield1  
 edinburg: edinburg1, ...  
 glasgow: glasgow, ...  
 belfast: belfast1, ...  
 exeter: exeter1, ...  
 londonderry: londonderry1, ...  
 cambridge: cambridge1

Domains :

- root domain = gb-station-loc
- second level domain = gb-location
- leaf domain = gb-stat-name

Binary Relations :

- gb-s-locations : gb-station-loc × gb-location
- gb-stat-names : gb-location × gb-stat-name

**great-britain**

england north-england: newcastle, lancaster, leeds, york  
 mid-england: sheffield, nottingham, birmingham, stoke  
 east-england: norwich, harwich, cambridge  
 south-england: london, dover, plymouth, exeter  
 wales north-wales: cardiff  
 south-wales: newport  
 scotland east-scotland: aberdeen, dundee, edinburgh  
 west-scotland: glasgow  
 northern-ireland northern-ireland1: londonderry, belfast

Domains :

- root domain = great-britain
- second level domain = gb-region
- third level domain = gb-subregion
- leaf level domain = gb-location

Binary Relations :

- gb-regions : great-britain × gb-region
- gb-subregions : gb-region × gb-subregion
- gb-locations : gb-subregion × gb-location

## Appendix II : BNF grammars

The BNF grammar for the deductive operations for classifications (the quoted braces are set delimiters) :

```

<class-expression> → <class-operation>
<class-operation> → <class-set>
<class-operation> → ROOT_CLASSES(<scope>)
<class-operation> → LEAF_CLASSES(<scope>)
<class-operation> → CLASSES(<scope>)
<class-operation> → SET_INTERSECTION(<class-operation>, <class-operation>)
<class-operation> → SET_DIFFERENCE(<class-operation>, <class-operation>)
<class-operation> → SET_UNION(<class-operation>, <class-operation>)
<class-operation> → IM_SUBCLASSES(<class>, <scope>)
<class-operation> → IM_SUPERCLASSES(<class>, <scope>)
<class-operation> → SUBCLASSES(<class>, <scope>)
<class-operation> → SUPERCLASSES(<class>, <scope>)
<class-operation> → UNION_OF_SUBCLASSES(<class-operation>, <scope>)
<class-operation> → UNION_OF_SUPERCLASSES(<class-operation>, <scope>)
<class-operation> → INTERSECTION_OF_SUBCLASSES(<class-operation>, <scope>)
<class-operation> → INTERSECTION_OF_SUPERCLASSES(<class-operation>, <scope>)
<class-operation> → DIFFERENCE_OF_SUBCLASSES(<class-operation>, <class-operation>,
<scope>)
<class-operation> → DIFFERENCE_OF_SUPERCLASSES(<class-operation>,
<class-operation>, <scope>)
<class-set>      →   "{"<class>{, <class>}*"}"
<scope>         →   "{"<basic-classification>{, <basic-classification>}*"}"

```

The BNF grammar for RA :

```

<rel-expression>  →   <rel-operation>
<rel-operation>  →   UNION(<rel-operation>, <rel-operation>)
<rel-operation>  →   INTERSECTION(<rel-operation>, <rel-operation>)
<rel-operation>  →   DIFFERENCE(<rel-operation>, <rel-operation>)
<rel-operation>  →   PRODUCT(<rel-operation>, <rel-operation>)
<rel-operation>  →   PROJECTION(<rel-operation>, <attribute set>)
<rel-operation>  →   RESTRICTION(<rel-operation>, <Boolean-predicate>)
<rel-operation>  →   JOIN(<rel-operation>, <rel-operation>, <Boolean-predicate>)
<rel-operation>  →   <basic relation name>
<attribute set>  →   "{"<attribute name>{, <attribute name>}*"}"
<Boolean-predicate> → <logical factor> {∨ <logical factor>}*
<logical factor>  →   {¬}0 <logical element> {∧ <logical factor>}*
<logical element> → <primitive> | (<Boolean-predicate>)
<primitive>      →   <attr-value-primit> | <attr-attr-primit> | <one-of-primit> |
<between-primit> | <truth-value>
<attr-value-primit> → <attribute name> <theta> <value>
<attr-attr-primit> → <attribute name> <theta> <attribute name>
<one-of-primit>   →   <attribute name> one-of(<value-set>)
<between-primit>  →   <attribute name> between(<numeric-value>, <numeric-value>)
<truth-value>    →   true | false
<theta>          →   < | • | = | • | > | •
<value-set>      →   "{"<value> {, <value>}*"}"
<numeric-value>  →   <integer> | <real number>

```

The BNF grammar for integrating RA and deductive expressions :

*Extensions to the above :*

<gen-expression> → <rel-expression> | <class-expression>

<class-operation> → <rel-expression>

*Replacements to the above :*

<class-set> → "{"<atomic-class>{, <atomic-class>}\*"}

<class> → <atomic-class> | <rel-expression> | <class-operation>

<numeric value> → <number> | <gen-expression>

<value> → <atomic value> | <gen-expression>

<value-set> → "{"<value>{, <value>}\*"}" | <class-operation>

## Appendix III : Expression evaluators

*Definition A1 :* Let Ex be any integrated expression where the outermost operation is a relational operation (i.e. *rel-expression*(Ex) = true), rdb the relational database, and CB the classification base. The function *rel-evaluate* is :

*rel-evaluate*(Ex, rdb, CB) =

*rn-rel*(rdb, Relname),

if Ex = Relname ∧ Relname ∈ { $\sigma_3$ (rel) | rel ∈ rdb} /\* the base case for a named relation of the rdb ; yields the relation \*/,

*restriction*(*rel-evaluate*(Ex1, rdb, CB), *normalize-pred*(P, rdb, CB)),

if Ex = RESTRICTION(Ex1, P),

*projection*(*rel-evaluate*(Ex1, rdb, CB), PA),

if Ex = PROJECTION(Ex1, PA),

*product*(*rel-evaluate*(Ex1, rdb, CB), *rel-evaluate*(Ex2, rdb, CB)),

if Ex = PRODUCT(Ex1, Ex2),

*join*(*rel-evaluate*(Ex1, rdb, CB), *rel-evaluate*(Ex2, rdb, CB), *normalize-pred*(JP, rdb, CB)),

if Ex = JOIN(Ex1, Ex2, JP),

*union*(*rel-evaluate*(Ex1, rdb, CB), *rel-evaluate*(Ex2, rdb, CB)),

if Ex = UNION(Ex1, Ex2),

*difference*(*rel-evaluate*(Ex1, rdb, CB), *rel-evaluate*(Ex2, rdb, CB)),

if Ex = DIFFERENCE(Ex1, Ex2),

*intersection*(*rel-evaluate*(Ex1, rdb, CB), *rel-evaluate*(Ex2, rdb, CB)),

if Ex = INTERSECTION(Ex1, Ex2),

*renaming*(*rel-evaluate*(Ex1, rdb, CB), RN),

if Ex = RENAMING(Ex1, RN).

*Definition A2 :* Let Ex be any integrated expression where the outermost operation is a class-operation (i.e. *class-operation*(Ex) = true), and rdb and CB as in the definition above. Let the function *scope-BCs*(Scope, CB) return the basic classifications given in Scope from CB (definition by-passed). The evaluator *class-evaluate* is :

*class-evaluate*(Ex, rdb, CB) =

{n<sub>1</sub>, ... n<sub>k</sub>} ,

```

        if Ex = {n1, ... nk} /* base case for a value set */,
        rel-class-set(rel-evaluate(Ex, rdb, CB) /* a base case, transforms a single
attribute relation to a set of classes */,
        if rel-expression(Ex)
        root-classes(scope-BCs(Scope, CB)) /* a base case */,
        if Ex = ROOT_CLASSES(Scope),
        leaf-classes(scope-BCs(Scope, CB)) /* a base case */,
        if Ex = LEAF_CLASSES(Scope),
        classes(scope-BCs(Scope, CB)) /* a base case */,
        if Ex = CLASSES(Scope),
        class-evaluate(Ex1, rdb, CB) ∪ class-evaluate(Ex2, rdb, CB),
        if Ex = SET_UNION(Ex1, Ex2),
        class-evaluate(Ex1, rdb, CB) ∩ class-evaluate(Ex2, rdb, CB),
        if Ex = SET_INTERSECTION(Ex1, Ex2),
        class-evaluate(Ex1, rdb, CB) - class-evaluate(Ex2, rdb, CB),
        if Ex = SET_DIFFERENCE(Ex1, Ex2),
        im_subclasses(value-conversion(Ex1, rdb, CB), scope-BCs(Scope, CB)),
        if Ex = IM_SUBCLASSES(Ex1, Scope),
        im_superclasses(value-conversion(Ex1, rdb, CB), scope-BCs(Scope, CB)),
        if Ex = IM_SUPERCLASSES(Ex1, Scope),
        subclasses(value-conversion(Ex1, rdb, CB), scope-BCs(Scope, CB)),
        if Ex = SUBCLASSES(Ex1, Scope),
        superclasses(value-conversion(Ex1, rdb, CB), scope-BCs(Scope, CB)),
        if Ex = SUPERCLASSES(Ex1, Scope),
        union_of_subclasses(class-evaluate(Ex1, rdb, CB), scope-BCs(Scope, CB)),
        if Ex = UNION_OF_SUBCLASSES(Ex1, Scope),
        union_of_superclasses(class-evaluate(Ex1, rdb, CB), scope-BCs(Scope, CB)),
        if Ex = UNION_OF_SUPERCLASSES(Ex1, Scope),
        intersection_of_subclasses(class-evaluate(Ex1, rdb, CB), scope-BCs(Scope, CB)),
        if Ex = INTERSECTION_OF_SUBCLASSES(Ex1, Scope),
        intersection_of_superclasses(class-evaluate(Ex1, rdb, CB), scope-BCs(Scope,
CB)),
        if Ex = INTERSECTION_OF_SUPERCLASSES(Ex1, Scope),
        difference_of_subclasses(class-evaluate(Ex1, rdb, CB), class-evaluate(Ex2, rdb,
CB), scope-BCs(Scope, CB)),
        if Ex = DIFFERENCE_OF_SUBCLASSES(Ex1, Ex2, Scope),
        difference_of_superclasses(class-evaluate(Ex1, rdb, CB), class-evaluate(Ex2, rdb,
CB), scope-BCs(Scope, CB)),
        if Ex = DIFFERENCE_OF_SUPERCLASSES(Ex1, Ex2, Scope).

```

**LEIKKEITÄ :**

The possibility for free mixing of deductive and RA operations means that the top level operation of a query may be a deductive operation. This is the case e.g. if one wants to find such gas pollutant families which contain pollutants whose air content has exceeded a certain critical limit L in January 1991 at the sensor station London1. The pollutants belonging to the family gas-pollutants are identified by the expression UNION\_OF\_SUBCLASSES({gas-pollutant}, {pollutant-families, pollutants}) which yields the set {COx, CO, CO2, NOx, NO, NO2, NO3, Ozone, O3, SOx, SO, ...}. The air-content for all of these is tested for exceeding the limit L at London1 in January 1991. The pollutant names for relevant pollutants are projected. Then the operation UNION\_OF\_SUPERCLASSES(PollSet, {pollutants}), where PollSet is the projection result, identifies the required pollutant families. Thus this query belongs to the integration case I. The sample expression **Q4** is given below and its result set is illustrated in Fig. 6.2.

**Sample Query Q4 :**

```
UNION_OF_SUPERCLASSES(
  PROJECTION(
    RESTRICTION(POLLUTANTS,
      (PStat = London1) ^ (TYear = 1991) ^
      (TMonth = jan) ^ (PMesment > L) ^
      (Ptant, one-of(
        UNION_OF_SUBCLASSES({gas-pollutant},
          {pollutant-families, pollutants}))))),
    {Ptant}),
  {pollutants})
```

{Ozone, SOx, CFC}
-------------------

**Fig. 6.2.** Sample result set for Query **Q4**