1999 Society for Design and Process Science
Printed in the United States of America

AUTOMATIC EXTRACTION OF DATABASE SCHEME SEMANTIC PROPERTIES USING KNOWLEDGE DISCOVERY TECHNIQUES

A. Bonifati, L. Palopoli, D. Saccà and D. Ursino
Dipartimento di Elettronica, Informatica e Sistemistica, Università della Calabria
Via Pietro Bucci, Rende (CS), Italy

This paper deals with the derivation of complex properties relating objects belonging to a set of database schemes and the exploitation of derived properties in query optimization and view maintenance problems. The derivation process has an iterative nature and works by case analysis over database schemes, in a way somehow similar to the way knowledge discovery processes behave on database extensions: for this reason, we call it Intentional Knowledge Discovery. Complex properties have a fuzzy nature and are represented and manipulated using a fuzzy variant of Description Logics. The paper presents detailed descriptions of procedures used to derive complex properties. As far as applications are concerned, the paper illustrates several query optimization cases and a view maintenance algorithm both based on the availability of extracted complex properties.

1. Introduction

The most appropriate exploitation of the enormous amount of data stored in electronic form poses nowadays new and challenging problems. These include issues ranging from the physical data level (e.g., proper data access structure design) to the purely conceptual one (e.g., database scheme integration). In particular, the presence, in the same operative environment, of database systems developed in different times and circumstances often makes it difficult to properly exploit available information. In such situations, a process must be carried out by which properties holding among objects belonging to the set of given database schemes are singled out and used to achieve an appropriate and unified description of available information.

The derivation of object properties from database schemes have recently attracted a growing interest in the research community. In particular, techniques have been developed that are capable to derive simple properties of database scheme objects, like, for instance, terminological properties (i.e., synonymies and homonymies) (Castano and Antonellis, 1997; Palopoli et al., 1998b), , inclusion properties (Fankhauser et al., 1991; Palopoli et al., 1998a) and meta-type mismatches (Palopoli et al., 1998c; Spaccapietra and Parent, 1994).

However, simple properties are not enough to capture several interesting aspects of scheme semantics, which are instead well represented as more complex formula denoting relationships holding between object patterns. In (Catari and Lenzerini, 1993) a logic is proposed to represent such complex relationships and its capabilities to properly capture scheme semantics is shown. In that paper, scheme complex properties are assumed to be provided by database experts. This assumption, however, is not easy to be met in application environments involving large numbers of databases. Indeed, in such contexts, there are so many scheme objects to be considered that manual analyses of scheme semantics turn out to be very hard to carry out. This is also due to the fact that some of the properties characterizing scheme semantics are not represented "in clear" but, rather, they remain hidden within scheme structures. Therefore, techniques are needed to "extract", in a fairly automatic fashion, useful relationships holding among scheme object patterns.

The problem of deriving useful knowledge from large amounts of (extensional) data has been successfully attacked by using Knowledge Discovery in Databases (hereafter, KDD) techniques (Fayyad, et al., 1996). Being able to extract useful information hidden within large data bunches have made KDD methods nowadays commonly applied with very encouraging results to many application contexts such as economics, biology, astronomy and so forth (Fayyad et al., 1996).

In order to extract complex properties from database schemes, in this paper we propose to adapt the knowledge discovery ideas to be applied for working on database scheme catalogues. As we focus on deriving knowledge from scheme catalogues, rather than from extensional data, we call our approach *Intentional Knowledge Discovery (IKDD,* for short*)*.

For representing and manipulating properties about scheme object patterns, we define and use a fuzzy variant of the language presented in (Catari and Lenzerini, 1993), that we call *Plausibility Description Logics* (hereafter, *PDL*). In our logics, each assertion is associated with a fuzzy coefficient stating assertion's plausibility. In our approach, PDL is used not only for representing scheme properties, but also for reasoning about them.

To derive complex properties, our techniques assume that simple properties about scheme objects, like those mentioned above, are available. Therefore, a pre-processing phase must be carried out that constructs suitable dictionaries storing such properties. Even if this paper mainly focuses on complex property derivation, in order to make it self contained, we also give some highlights on the techniques devoted to deriving simple properties about scheme objects; these techniques are presented in details in (Palopoli et al., 1998a, b, c).

Extracting properties about object patterns in the form of PDL assertions have several applications in the database context, including the development of tools for supporting the integrated access to Cooperative Information Systems (Papazoglu et al., 1992; Catarci, 1993; Wiederhold, 1992;Ullman, 1997; Levy, 1996; Garcia-Molina et al., 1997), query optimization (Chaudhuri et al., 1995), view materialization (Hanson, 1987) and data warehousing. In this paper, we concentrate on describing query optimization and view materialization applications. In particular, some query optimization patterns are described for which the availability of suitable PDL assertions allows to greatly simplify query answering. Furthermore, we provide an algorithm able to automatically implement a view materialization policy that is based on the availability of PDL assertions about schemes which views are defined upon.

The plan of the paper is as follows. In Section 2 we briefly overview Knowledge Discovery in Databases and introduce the Plausibility Description Logics. Section 3 provides a general overview of our approach. The pre-processing phase is described in Section 4. The core complex property extraction phase is illustrated into details in Section 5. Section 6 is devoted to present the application of IKDD to query optimization and view materialization. Finally, in Section 7, we draw our conclusions.

2. Intentional Knowledge Discovery in Databases

2.1. Knowledge Discovery in Databases

The enormous growth of data stored in database systems has probably nowadays outpaced the capabilities of available techniques to support appropriate data analysis. In this contexts, traditional methods, mainly based on humans dealing directly with rough data, cannot be successfully applied to extract and comprehend useful information from huge amounts of data. Standard reporting capabilities of DBMS also fail in supporting such complex analysis activities. Therefore, the necessity has arisen of *intelligent* tools for *automated* knowledge extraction. The area of *Knowledge Discovery in Databases* deals with designing such tools (Fayyad et al., 1996).

By "Knowledge Discovery in Databases" we mean the non-trivial process of identifying valid, novel, potentially useful, and, ultimately, understandable patterns in data (Fayyad et al., 1996). Often the term *Data Mining* is used in the place of KDD; actually the two terms are not synonyms in that Data Mining is only one of the steps of the Knowledge Discovery Process. More in particular Data Mining is the step of the KDD process which extracts patterns from data; the process of KDD includes also many other phases, such as pre-processing, pattern validation etc.

The KDD process is a semi-automatic, interactive and iterative process, composed by numerous steps; the intervention of the end user is required in many of them. More in details we can distinguish the following steps (Fayyad, et al., 1996):

- Understanding the application domain, the existing knowledge and user goals,
- Creating a set of target data,
- Cleaning and pre-processing data for taking the possible presence of noise into account, for managing missing data fields, etc,
- Reducing data by selection and projection operations,
- Choosing the Data Mining task,
- Choosing the Data Mining algorithms,
- Executing the Data Mining algorithms for extracting patterns of interest,
- Interpreting extracted patterns,
- Consolidating discovered knowledge incorporating it in the known data and solving potential conflicts with previous knowledge.

The KDD process can require some iterations of these steps. It appears clear that, even if the core step is the Data Mining, the other steps are fundamental for a successful application of KDD techniques. For more details about KDD, see (Fayyad et al., 1996).

Our approach is based on the idea of adopting the concepts of knowledge discovery to database catalogues, in order to discover interesting properties consisting of complex relationships holding between scheme objects. In other words, properties are extracted from database schemes applying a process which can be looked at as a special KDD process. Contrary to traditional knowledge discovery techniques, our methods works mainly on schemes rather than on extensional data[1]. We have coined the term *Intentional Data Mining* to refer to this process of extracting new information from database schemes. The associated Knowledge Discovery process is similarly called *Intentional Knowledge Discovery in Databases (IKDD).*

2.2. Plausibility Description Logic: A Language for IKDD

In order to reason about properties of complex data patterns, we use a variant of Description Logics (DL), a logic developed to represent and model complex knowledge. The variant we define and use here,

[1] Even if, in several application contexts, when multiple and heterogeneous data sources are involved, the sharp distinction traditionally made in databases between schemes and data looses much of its importance (Abiteboul, 1997).

based in part on the DL language of (Catari, 1993), is called Plausibility Description Logic (PDL). We implicitly assume that our input database schemes are represented in PDL. This is not an actual limitation, since translations to PDL exist from all data models (Catari, 1993). This section is devoted to the presentation of the Plausibility Description Logic.

2.2.1. *Preliminary Definitions*

In our model the universe of discourse is partitioned into an *instance level* and a *class level*.

Basic elements of the instance level are *objects*; these are atomic elements and are represented through a unique symbol identifying each of them. Generally, objects are grouped into *classes*; objects of a class C form the set of *instances* of C.

Associations between objects are expressed by grouping them into *tuples*. The number of objects composing a tuple is called the *arity* of the tuple. A *relationship* is a set of tuples of the same arity; tuples belonging to a relationship R form the set of *instances* of R.

The class level, specified using a scheme, consists of a group of class symbols, called *alphabet*, and by some specifications about how classes are related to each other. More precisely, an alphabet is a list of symbols; these can be entities, relationships, roles, attributes, values and domains.

An *entity* is an abstraction of a class of objects; objects constitute istances of the class. Properties of an entity are determined by its attributes and by relationships with other entities.

As previously mentioned, a *relationship* is a class of object tuples of the same arity; this is also the arity of the relationship. Tuples form the set of instances of the relationship.

A *role* represents a component of a relationship; therefore the number of roles of a relationship is equal to its arity. Each class participating in a relationship is associated with a role of the relationship and is called *filler* of this role. For example the relationship Teaches has two roles: Teacher and Student. Each tuple of the relationship has one component for each role of the corresponding relationship.

A *domain* is a set of values. Examples of domains are integers, strings, dates, reals etc. An *attribute* is a named relationship linking an entity or a relationship with a domain.

2.2.2. *Syntax*

The language is based on an alphabet B of symbols including class names, the special symbols $\top, \bot, \sqcap, \sqcup, \exists, \forall$ and usual parentheses.

A *class expression* is either an entity expression or a relationship expression. An *entity expression* over the alphabet B is constructed according to the following rules:

$$\begin{aligned} C,\ F \rightarrow\ & E \mid \\ & C \sqcup F \mid \\ & C \sqcap F \mid \\ & \neg\ C \mid \\ & \forall R[U].T_1{:}C_1, \ \ldots, T_n{:}C_n \mid \\ & \exists R[U].T_1{:}C_1, \ \ldots, T_n{:}C_n \mid \\ & \forall\ A.D \mid \\ & \exists\ A.D \end{aligned}$$

where C, F and E are entity expressions, R is a relationship symbol and $T_1, \ldots, T_n, U$ are role symbols from B.

A *relationship expression* is a formula of the form

$$R[U_1, U_2, \ldots, U_n]$$

where *R* is a relationship symbol over the alphabet *B* and $\{U_1, U_2, \ldots, U_n\} = rol(R)$ are the roles associated with *R*.

An *assertion* is a statement of the form: $L_1 \sqsubseteq L_2$ where L_1 and L_2 are class expressions of the same type.

2.2.3. Semantics

Semantics of PDL is based on interpretations. An *interpretation* $I = (\Delta^I, \bullet^I)$ consists of:

1. a non empty set Δ^I, called *universe of I*, which comprises all the objects;
2. a mapping $\bullet^I$, called *interpretation function* of I.

For each *I*, the interpretation function of *I* assigns to each *entity expression* a subset of Δ^I, according to the following rules:

- $T^I = \Delta^I$
- $\bot^I = \varnothing$
- $(C \sqcup F)^I = C^I \cup F^I$
- $(C \sqcap F)^I = C^I \cap F^I$
- $(\neg C)^I = \{a \in \Delta^I \mid a \notin C^I\}$
- $(\forall R[U].T_1{:}C_1, \ldots, T_n{:}C_n)^I = \{a \mid \forall r \in R.\ (r[U] = a) \Rightarrow$
  $(r[T_1] \in C_1^I \wedge \ldots \wedge r[T_n] \in C_n^I)\}$
- $(\exists R[U].T_1{:}C_1, \ldots, T_n{:}C_n)^I = \{a \mid \exists r \in R^I.\ (r[U] = a) \wedge$
  $(r[T_1] \in C_1^I \wedge \ldots \wedge r[T_n] \in C_n^I)\}$
- $(\forall A.D)^I = \{a \mid \forall\ (a,b) \in A^I.\, b \in D^I\}$
- $(\exists A.D)^I = \{a \mid \exists\ (a,b) \in A^I.\, b \in D^I\}$

For each *I*, the interpretation function of *I* assigns a set of labelled tuples to each *relationship expression* as:

$$(R[U_1, U_2, \ldots, U_n])^I = R^I$$

where, if *R* is a relationship with roles $\{U_1, U_2, \ldots, U_n\}$, R^I is a set of labelled tuples of the form

$$< U_1{:}u_1, \ldots, U_m{:}u_m >$$

and $u_1, \ldots, u_m \in \Delta^I$. In the following, if *r* is an instance of *R*, we shall use $r[U_i]$ to denote the object associated to U_i by *r*.

The semantic of the *assertion* $L_1 \sqsubseteq L_2$ is given as follows:

- if L_1 and L_2 are entity or attribute expressions, the assertion is satisfied if $L_1^I \subseteq L_2^I$;
- if $L_1 = R_1[U_1, \ldots, U_n]$ and $L_2 = R_2[T_1, \ldots, T_m]$ are relationship expressions, then the assertion is satisfied if $m = n$ and for each tuple $<U_1{:}d_1, \ldots, U_n{:}d_n>$ in R_1^I, the tuple $<T_1{:}d_1, \ldots, T_n{:}d_n>$ is in R_2^I.

An interpretation *I* is called a *model* of a set of assertions Σ if each assertion in Σ is satisfied by *I*.

2.2.4. *Plausibility Coefficients*

Differently from standard DL, in our Plausibility Description Logic, assertions are associated with a plausibility coefficient. Therefore, PDL assertions have a truth value measured as a real number between 0 and 1, that measures their plausibility. An assertion $L_1 \sqsubseteq L_2$, with plausibility coefficient f, will be represented as a triplet $<< L_1, L_2, f >>$.

3. General Description of the Approach

The algorithm for extracting complex patterns from a set of database schemes takes in input a list of database schemes $S = S_1 \ldots S_n$, a dictionary of lexicographic synonymy properties *LSPD* and a (possibly empty) dictionary of supplied inclusion properties *SIPD*. The *LSPD* stores triplets of the form $[A, B, f]$, where A and B are the involved objects and f is a fuzzy coefficient denoting the plausibility for a lexical synonymy to hold between them. The *SIPD* contains triplets of the form $<A,B,f>$, where A is the included object, B is the including object and f is a fuzzy coefficient which expresses the strength of the assertion. Triplets stored in the *SIPD* represent, generally, inclusion properties between objects of the same scheme (*intrascheme properties*); some further inclusion properties holding between objects belonging to different schemes can be possibly stored in this dictionary by the DBAs. The algorithm provides in output a dictionary *CAD* of assertions holding between complex object patterns.

The algorithm exploits some support dictionaries, that are:

- A *Synonymy Property Dictionary SPD,* which stores synonymies between scheme objects. These indicate that two objects have different names but the same meaning within their schemes. Synonymies are represented by triplets of the form $\lfloor A, B, f \rfloor$, where A and B are the involved objects and f is a fuzzy coefficient expressing the strength of the property.
- An *Homonymy Property Dictionary HPD,* storing homonymies between scheme objects. These indicate that the two objects have the same name but different meanings within their respective schemes. Homonymies are denoted by triplets of the form $\|A, B, f\|$, where A and B are the involved objects and f is a fuzzy coefficient.
- An *Inclusion Property Dictionary IPD*, which stores triplets of the form $< A, B, f>$, where A is the included object, B is the including object and f is a fuzzy coefficient expressing the plausibility of the property. They differ from assertions of *SIPD* because *(i)* they are not provided by a DBA but constructed in the Pre-processing phase of our method (see below Section 4) and, *(ii)* they involve objects belonging to different schemes.
- A *Type Conflict Dictionary TCD,* storing type conflicts; a type conflict between two objects denotes that they represent the same concept, yet having different types within their schemes (e.g., one is an entity and the other is a relationship, or one is an attribute and the other is an entity, and so on). The Type Conflict Dictionary stores triplets of the form $\lceil A, B, f \rceil$, where A and B are the involved objects and f is a fuzzy coefficient expressing type conflict plausibility.

The output of the algorithm is a dictionary *CAD* storing assertions holding between complex object patterns. One such assertion can be represented by a triplet of the form, $<< A, B, f >>$, where A and B are DL formulae. Its meaning is that the assertion A ... B holds with plausibility f. Thus, complex assertions are inclusion properties; they differ from properties stored in the *IPD* because these latter ones involve simple entity symbols whereas complex assertions involve object patterns represented as DL formulae of any complexity.

Our method consists of two phases: during the first phase database schemes are analyzed for extracting synonymies, homonymies, inclusions, type conflicts and for normalizing schemes. No complex assertion is derived during the first phase; these are instead derived in the second phase.

Our main procedure is as follows:

```
Algorithm for extracting complex patterns
Input:  a list of database schemes $S = S_1, \dots, S_n$; a dictionary LSPD of
        lexicographic synonymies; a dictionary SIPD of Supplied Inclusion
        Properties;
Output: a dictionary CAD of assertions between complex patterns;
Var
   SPD: Synonymy Property Dictionary;
   IPD: Inclusion Property Dictionary;
Begin
   Pre-processing (S,LSPD,SIPD,SPD,IPD);
   Knowledge_Extraction (S,SPD,IPD,CAD)
End
```

4. Pre-processing

For the sake of completeness, in this section, we illustrate the main issues associated with the pre-processing phase, whose detailed presentation can be found in (Palopoli, 1998a,b,c).

The procedure Preprocessing (S,LSPD,SIPD,SPD,IPD) takes in input a set of schemes, derives some interscheme properties and then normalizes schemes. More in particular, the procedure iteratively derives basic nominal properties first, (i.e. synonymies and homonymies); then, it derives inclusions and some further synonymies; finally type conflicts are detected. These steps are repeated until no new valid property is derived. A property is valid if its plausibility coefficient is large enough. After that property extraction has taken place, schemes are normalized. In particular, non-binary relationships are converted into binary ones and type conflicts are resolved:

```
Procedure Preprocessing (var S: a list of database schemes; LSPD: a
Lexicographic
          Synonymy Property Dictionary; SIPD: a Supplied Inclusion Property
          Dictionary; var SPD: a Synonymy Property Dictionary; var IPD: an
          Inclusion Property Dictionary)
var
   HPD: an Homonymy Property Dictionary;
   TCD: a Type Conflict Dictionary
Begin
   repeat
      Derive_Basic_Nominal_Properties(S,LSPD,SPD,HPD);
      Derive_Related_Syn_Inc_Properties(SIPD,SPD,IPD);
      Derive_Type_Conflict(S,LSPD,SPD,HPD,TCD)
   until no further valid property is derived;
   NBR_Normalization(S,SPD,HPD,TCD);
   TC_Normalization(S,SPD,HPD,TCD)
End
```

In the following subsections, we provide some highlights about the procedures listed above.

4.1.Deriving Basic Nominal Properties

The procedure for extracting basic nominal properties derives synonymies and homonymies between objects belonging to different schemes. For each scheme object (entity or relationship), our algorithm considers its structure, i.e. its attributes and its context (Castano and Antonellis, 1997). The term "context" here indicates attributes, entities, generalization hierarchies and relationships involving a given object in the scheme. Exploiting the context is motivated by the consideration that entities having the same real world semantics are very often characterized by the presence of common elements in their context. Moreover, it is generally accepted that similarity techniques based on hypernimy and synonymy relationships between concepts are more precise than techniques solely based on the attribute analysis (Castano and Antonellis, 1997; Palopoli 1998b,c). Finally, the algorithm takes into account the relevance of attributes in distinguishing the semantics of entities/relationships (Fankhauser et al., 1991). For example the attribute *Surname* is more relevant than the attribute *Identifier* for distinguishing the entity *Person* from the entity *Car*. The procedure for deriving basic nominal properties is the following:

```
Procedure Derive_Basic_Nominal_Properties(S: a list of database schemes; LSPD:
a
          Lexicographic Synonymy Property Dictionary; var SPD: a Synonymy
Property
          Dictionary; var HPD: an Homonymy Property Dictionary)
Begin
   Derive_Rough_E_Syn(S,LSPD,SPD);
   Derive_R_Syn(S,LSPD,SPD);
   Derive_Refined_E_Syn_Hom(S,LSPD,SPD,HPD);
   Select_Strong(S,SPD,HPD)
End
```

The procedure derives first the so called *rough* synonymies between entities, i.e. synonymies resulting by taking into account only their structure. Then, synonymies between relationships are derived by taking into account both their structure and their context. Then, the so called "refined" synonymies and homonymies between entities are derived taking into account also entity contexts. A filtering step finally discards properties whose plausibility coefficient is under a certain threshold (since they are assumed not to be valid). A complete description of this procedure is beyond the purpose of this paper. Interested readers are referred to (Palopoli, 1998b) for details.

4.2. Deriving Related Synonymy and Inclusion Properties

The procedure *Derive_Related_Syn_Inc_Properties*(*SIPD,SPD,IPD*) derives further synonymy properties and inclusion ones. We call them *related* because they cannot be derived independently from one another; indeed the derivation of an inclusion property leads to the derivation of a synonymy property and vice versa.

Both synonymy and inclusion properties extracted in this second phase can be classified as *always deducible* (A-properties) and *conditionally deducible* (C-properties). A-properties are derived independently from the values of involved coefficients, whereas the derivability of C-properties depends on them (Palopoli 1998a,b).

The algorithm uses an associative network (Fankhauser et al., 1991), that is, a labelled graph whose nodes represent scheme objects and whose edges represent properties between objects. Edge labels denote property types and strength. Properties already included in *SPD* and *SIPD* are represented as edges in the network. New properties are then derived by constructing suitable closures over the network.

Inferred properties must be checked for possible conflicts with old ones, so a validation phase is necessary. This is conducted in part automatically but may also require the intervention of human experts. In particular, some derived properties may result in contradiction with the belief of some expert, in which case a negotiation phase takes place between the system and the expert; during this phase the system provides inference tracing justifying the deduction of the contradictory piece of information and the expert supplies further information either validating or modifying or rejecting the assertions included in the inference tracing. Finally, a suitable *SPD* or *IPD* entry is created for each derived validated edge of the associative network. More formally, the procedure is as follows:

```
Procedure Derive_Related_Syn_Inc_Properties(SIPD: a Supplied Inclusion
          Property Dictionary;  var SPD: a Synonymy Property Dictionary; var
IPD: an
          Inclusion Property Dictionary)
Var
   Net: an associative network;
Begin
   Dictionaries_To_Network(Net,SPD,SIPD);
   Compute_A_Properties(Net);
   Compute_C_Properties(Net);
   Validation(Net);
   Net_To_Dictionaries(Net,SPD,IPD)
End
```

More details about this procedure can be found in (Palopoli, 1998a,b).

4.3. Deriving Type Conflicts

The procedure *Derive_Type_Conflicts*(*S,LSPD,SPD,HPD,TCD*) derives type conflicts arising when the same concept is represented, in different schemes, by objects of different types (e.g. one is an entity and the other is a relationship or one is an attribute and the other is an entity, etc.). The procedure starts by extracting conflicts between an attribute of an entity and an entity. Then, it discovers conflicts between an attribute of a relationship and an entity. Finally, conflicts between entities and relationships are derived. Conflicts between an attribute and a relationship are not considered (Palopoli et al., 1998c) since such conflicts actually reduce to conflicts between an attribute and an entity linked to the relationship and, therefore, they can be solved as such (Batini and Lenzerini, 1984). The procedure follows:

```
Procedure Derive_Type_Conflicts(S: a list of database schemes; LSPD: a
          Lexicographic Synonymy Property  Dictionary; SPD: a Synonymy Property
          Dictionary; HPD: an Homonymy Property Dictionary; var TCD: a Type
          Conflict Dictionary)
Begin
   Compute_EAttribute_Entity_Conflicts(S,LSPD,SPD,HPD,TCD);
   Compute_RAttribtue_Entity_Conflicts(S,LSPD,SPD,HPD,TCD);
   Compute_Entity_Relationship_Conflicts(S,LSPD,SPD,HPD,TCD)
End
```

Details about type conflict derivation can be found in (Palopoli, 1998c).

4.4. Scheme Normalizations

The procedure *NBR_Normalization(S,SPD,HPD,TCD)* transforms each non-binary relationship into a set of binary ones. Each of these transformations modifies schemes and dictionaries.

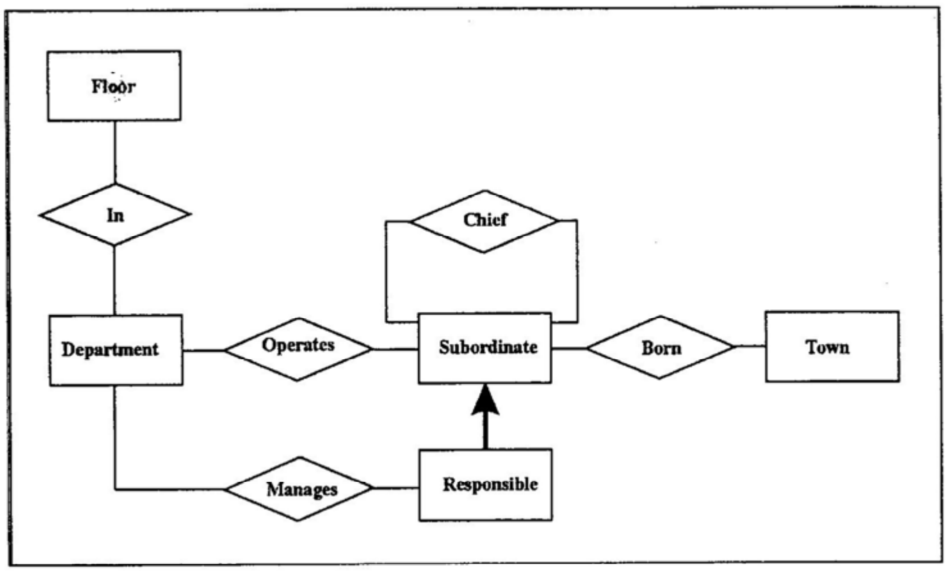

Fig. 1 Scheme PD: the Production Department Database.

The procedure *TC_Normalization(S,SPD,HPD,TCD)* normalizes a set of schemes solving type conflicts; it implements the methodology proposed in (Batini et al., 1984).

4.5. An Example Case

Consider the schemes in Figure 1 and Figure 2, representing the Production (denoted PD) and Administration (denoted AD) departments of an organization, respectively.

The pre-processing phase derives the following properties (Palopoli, 1998b)[2]:

$\lfloor \text{Department}_{[PD]}, \text{Division}_{[AD]}, 0.57 \rfloor$ $\lfloor \text{Town}_{[PD]}, \text{Birthplace}_{[AD]}, 0.98 \rfloor$
$\lfloor \text{Born}_{[PD]}, \text{Born}_{[AD]}, 0.97 \rfloor$ $\lfloor \text{Subordinate}_{[PD]}, \text{Employee}_{[AD]}, 0.74 \rfloor$
$\lfloor \text{Operates}_{[PD]}, \text{Works}_{[AD]}, 0.57 \rfloor$ $\lfloor \text{Chief}_{[PD]}, \text{Manager}_{[AD]}, 0.73 \rfloor$
$\langle \text{Engineer}_{[AD]}, \text{Subordinate}_{[PD]}, 0.35 \rangle$ $\langle \text{Responsible}_{[PD]}, \text{Employee}_{[AD]}, 0.21 \rangle$

5. Knowledge Extraction

This section is devoted to illustrating in details the steps executed within our core phase 2. The procedure *Knowledge_Extraction(S,SPD,IPD,CAD)* extracts properties involving, in general, complex class expressions. The extraction of complex properties is done in two steps:

- *Step 1*, where most interesting classes are singled out on the basis of a weight assigned to them by an algorithm which uses dictionaries constructed in the pre-processing phase;
- *Step 2*, in which new properties, involving interesting classes identified in Step 1, are derived.

The procedure is as follows:

[2] Here and in the following we denote by $O_{[S]}$ the object O of the scheme S.

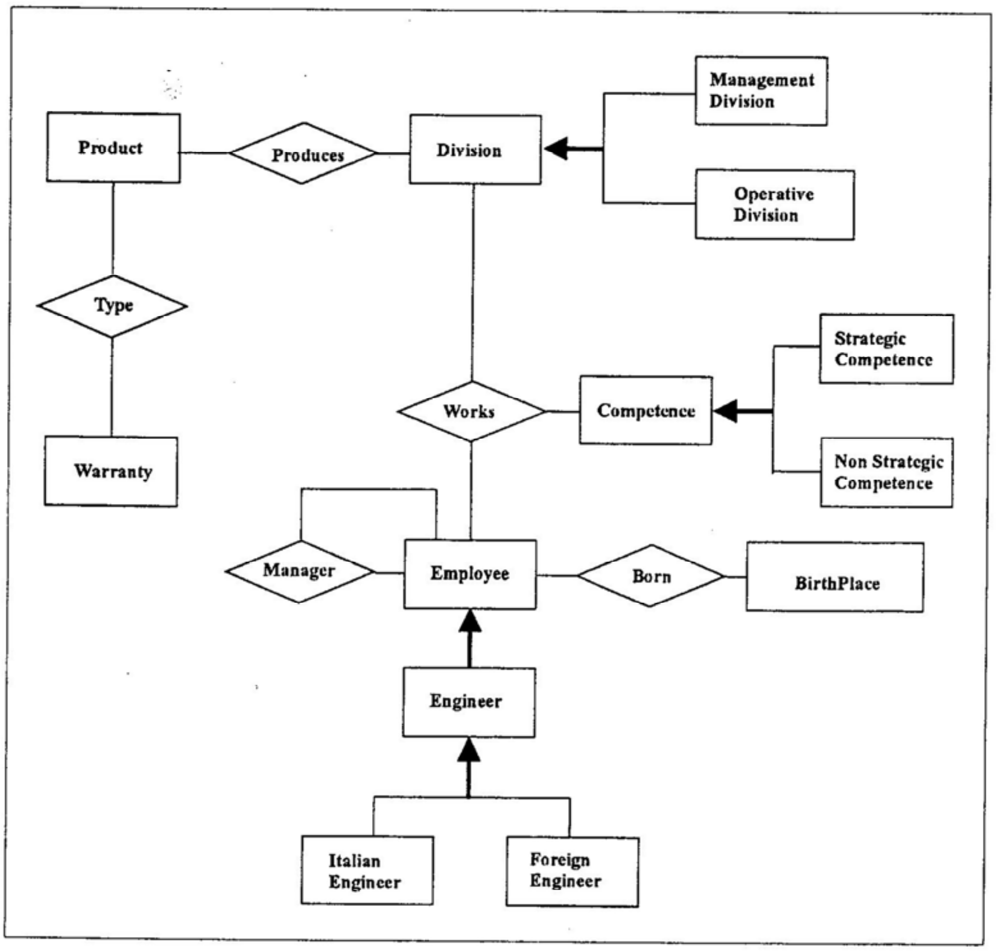

Fig. 2 Scheme AD: the Administration Department Database.

```
Procedure Knowledge_Extraction(S: a list of database schemes; SPD: a Synonymy
          Property Dictionary; IPD: an Inclusion Property Dictionary;  var CAD:
a
          dictionary of assertions between complex patterns)
Begin
   Select_Interesting_Classes(S,SPD,IPD,CAD);
   CP_Extraction(CAD)
End
```

The two procedures listed above are illustrated in details in the following two subsections.

5.1. Selecting Interesting Classes

This procedure exploits the information stored in the synonymy and inclusion dictionaries in order to associate an interest weight to entities and relationships of each input scheme. The rationale here is that in general, there may exist a virtually infinite number of properties which could be extracted; therefore it is crucial to single out most relevant objects so that only the properties regarding them are extracted. In order to single out interesting entities, an interest threshold value is computed *for each entity*; therefore, all scheme entities with interest coefficient greater than the threshold are considered *interesting*. The selection of interesting relationships is performed analogously. Using the set of interesting scheme objects we have selected, we obtain an initial set of assertions to include in the *CAD*, as explained later in this section. The procedure is as follows:

```
Procedure Select_Interesting_Classes(S: a list of database schemes; SPD: a
Synonymy
          Property Dictionary;  IPD: an Inclusion Property Dictionary;  var
CAD: a
          dictionary of assertions between complex patterns)
Var
   ID: An Interest Dictionary;
Begin
   ID := Ø ;
   for each scheme $S_k \in S$ do begin
      for each entity $E_i \in S_k$  do begin
         Compute_E_Interest_Coefficient(SPD,IPD,$E_i$,IC);
         ID := ID ∪ ($E_i$,IC)
      end;
      for each relationship $R_j \in S_k$  do begin
         Compute_R_Interest_Coefficient(SPD,IPD,$R_j$,IC);
         ID := ID ∪ ($R_j$,IC)
      end
   end;
   Discard_Weak_E(ID);
   Discard_Weak_R(ID);
   Populate_CAD(ID,IPD,CAD)
End
```

In *Select_Interesting_Classes*(*S,SPD,IPD,CAD*), several procedures are called, which are presented in the following subsections.

5.1.1. Computing an interest coefficient for entities and relationships

The procedure *Compute_E_Interest_Coefficient(SPD,IPD,E_i,IC)* computes an interest coefficient *IC* for the entity E_i. The interest coefficient associated with the entity is computed as e_j *Val(Rj)* where the R_j 's represent relationships directly connected to E_i and *Val(R_j)* takes into account how many times *Rj* is involved in interscheme properties and what is the plausibility coefficient value of these properties.

More precisely, *Compute_E_Interest_Coefficient(SPD,IPD,E_i,IC)* is implemented as follows:

```
Procedure Compute_E_Interest_Coefficient(SPD: a Synonymy Property Dictionary;
          IPD: an Inclusion Property Dictionary; $E_i$: an entity; var IC: Real)
Var
   $R_k$ : a relationship;
Begin
   IC := 0 ;
   for each relationship $R_k$ directly connected to $E_i$ do begin
      Compute_R_Val(SPD,IPD,$R_k$,Val);
       IC := IC + Val
   end
End
```

The procedure `Compute_R_Val(SPD,IPD,`R_k`,Val)` returns a value encoding a "local" interest

of the
proper
will b

```
Pro
Var
Beg
f
```

The
IC f
Compu

5.1.2.

The
thresho
consid

```
Proc
Cons
   D
Var
   T
   E
Begi
   C
   C
   T
   f
End
```

In th
the mo

of the relationship R_k. The underlying assumption here is that the more a relationship is involved in properties appearing in dictionaries associated with high factors, the more probable it is that this relationship will be used to extract new interesting properties from schemes:

```
Procedure Compute_R_Val(SPD: a Synonymy Property Dictionary;  IPD: an Inclusion
          Property Dictionary; $R_k$ : a relationship; var Val: Real)
Var
   t: a tuple in SPD;
   t': a tuple in IPD;
Begin
   Val := 0 ;
   for each t $\in$ SPD do
      if t = $\lfloor R_k, O, f \rfloor$ or t = $\lfloor O, R_k, f \rfloor$ then
         if <$R_k$, O, g> $\in$ IPD or <O, $R_k$, g> $\in$ IPD then
            Val := Val + Max(f,g)
         else
            Val := Val + f;
   for each t' $\in$ IPD do
      if t' = <$R_k$, O, g > or t' = <O, $R_k$, g > then
         if $\lfloor R_k , O , f \rfloor \notin$ SPD and $\lfloor O , R_k , f \rfloor \notin$ SPD then
            Val := Val + g
```

The procedure *Compute_R_Interest_Coefficient(SPD,IPD,R_j,IC)* computes an interest coefficient *IC* for the relationship R_j in a way that is very similar to the way *Compute_E_Interest_Coefficient(SPD,IPD,E_j,IC)* proceeds and, therefore, it is not further illustrated.

5.1.2. Discarding Uninteresting Entities and Relationships

The procedure *Discard_Weak_E(ID)* discards all uninteresting entities. To this purpose, an interest threshold value is computed. All entities associated with interest coefficients lesser than the threshold are considered uninteresting. The procedure follows:

```
Procedure Discard_Weak_E (var ID: an Interest Dictionary)
Const
   $D_R$ = 3;
Var
   $Th_E$: Real;
   $E_{Max}$, $E_{Min}$: Real;
Begin
   Compute_E_Max(ID,$E_{Max}$);
   Compute_E_Min(ID,$E_{Min}$);
   $Th_E$ := ($E_{Max}$ + $E_{Min}$)/$D_R$;
   for each tuple ($E_i$,$Val_i$)$\in$ID such that $E_i$ is an entity and $Val_i$ < $Th_E$ do
      ID := ID - {($E_i$,$Val_i$)}
End
```

In this procedure, D_R is a normalization factor used to tune up the threshold: the smaller its value is, the more selective the threshold will be. The procedure *Compute_E_Max(ID,E_{Max})* (resp., *Compute_E_Min(ID,E_{Min})*) returns the maximum (resp., the minimum) coefficient associated to entities in the *ID*. The procedure *Discard_Weak_R(ID)* is analogous to this one, and will not be, therefore, presented in details.

5.1.3. Populating the CAD with an initial set of assertions

The procedure *Populate_CAD(ID,IPD,CAD)* populates the CAD with an initial set of assertions. These assertions consist essentially in:

- the most interesting properties of the *IPD*,
- some new assertions between complex patterns involving objects of a single scheme.

In order to associate the new assertions with proper inclusion coefficients, we proceed by first asking the database expert to state which of these assertions are meaningful and then by submitting a suitable aggregate query to compute the coefficients associated to meaningful assertions (note that both the left-hand and the right-hand sides of PDL assertions correspond to simple queries relative to a single database). Procedure Populate_CAD(ID,IPD,CAD)is implemented as follows:

```
Procedure Populate_CAD (ID: an Interest Dictionary; IPD: an Inclusion Property
          Dictionary; var CAD: a dictionary of assertions between complex
patterns)
Var
   TCAD: a (temporary) dictionary of assertions between complex patterns;
Begin
   CAD := $\emptyset$;
   TCAD := $\emptyset$;
   for each tuple $(O_i,Val_i) \in ID$ do begin
      if $O_i$ is an entity then
         for each tuple $t \in IPD$ such that $t=<O_i,O_j,f>$ or $t=<O_j,O_i,f>$ then
            CAD := CAD $\cup$ t;
      if $O_i$ is a relationship then begin
         Let $T_1,\ldots,T_n$ be the roles of $R_i$ and $C_1,\ldots,C_n$ be the entities connected
to
         $R_i$;
         TCAD := TCAD $\cup$ << $\exists R_i[T_1].T_2:C_2,\ldots,T_n:C_n, C_1, f_{\exists 1}$ >> $\cup$
                             << $\forall R_i[T_1].T_2:C_2,\ldots,T_n:C_n, C_1, f_{\forall 1}$ >> $\cup$
                             ...
                             << $\exists R_i[T_n].T_1:C_1,\ldots,T_{n-1}:C_{n-1}, C_n, f_{\exists n}$ >> $\cup$
                             << $\forall R_i[T_n].T_1:C_1,\ldots,T_{n-1}:C_{n-1}, C_n, f_{\forall n}$ >>
      end
   end
   Validate_TCAD(TCAD);
   CAD := CAD $\cup$ TCAD
End
```

Initially, values $f_{\exists 1}, f_{\forall 1}, \ldots, f_{\exists n}, f_{\forall n}$ are undefined. The procedure *Validate_TCAD(TCAD)* is an interactive procedure allowing the database expert to specify which of the assertions currently stored in *TCAD* are meaningful[3]. In addition, for each validated assertion, *Validate_TCAD(TCAD)* submits suitable aggregate queries on the associated databases in order to compute the proper plausibility coefficient. An example will help in clarifying this issue.

5.1.4. Example

Consider again the schemes reported inFigure1and in Figure 2. The procedure *Select_Interesting_Classes* singles out as interesting the relationships $Works_{[AD]}$, $Born_{[PD]}$, $Born_{[AD]}$, $Operates_{[PD]}$, $Produces_{[AD]}$ and $Manages_{[PD]}$ and the entities $ForeignEngineer_{[AD]}$, $ItalianEngineer_{[AD]}$, $Engineer_{[AD]}$, $OperativeDivision_{[AD]}$, $Division_{[AD]}$, $Competence_{[AD]}$, $Responsible_{[PD]}$, $Subordinate_{[PD]}$, $Employee_{[AD]}$. Two of the assertions tentatively selected to populate the dictionary are:

$$<< \exists Born_{[AD]}[IN].NL{:}Birthplace_{[AD]},\ Employee_{[AD]},\ g >>$$
$$<< \exists Born_{[AD]}[NL].IN{:}Employee_{[AD]},\ Birthplace_{[AD]},\ f >>$$

For instance, the former assertion denotes the subset of employees born in at least one birthplace; it does not appear meaningful and will be probably discarded by the DBA. The latter assertion denotes the subset of birthplaces where at least an employee was born and appears semantically meaningful, and, thus, will not be discarded. Therefore, a plausibility factor must be associated to the latter assertion. To this purpose, the following aggregate intrascheme queries can be executed (SQL has been used for simplicity):

```
SELECT COUNT (DISTINCT E.BIRTHPLACE_CODE) AS A_ATT
INTO A_REL
FROM EMPLOYEE E

SELECT COUNT (DISTINCT E.BIRTHPLACE_CODE) AS B_ATT
INTO B_REL
FROM BIRTHPLACE B

SELECT A_ATT/B_ATT
FROM A_REL,B_REL
```

The query returns the ratio of the number of birthplaces where at least an employee was born over the total number of birthplaces. This ratio could be assumed as the plausibility factor of the assertion. However, since the plausibility factor has a fuzzy measure, the DBA is asked to validate or possibly tune it.

5.2. Pattern Extraction

The procedure *CP_Extraction(CAD)* implements the second step of the intentional knowledge discovery process. As already stated, the general form of PDL assertions extracted by our method correspond to formulae $L_1 \sqsubseteq L_2$ to which a plausibility factor f is associated. Here, both L_1 and L_2 are Description Logic class expressions. The pre-processing phase extracts properties where both L_1 and L_2 are simple entity symbols. The method we are presenting next derives properties involving more complex expressions. The procedure works by case analysis and is iterated until to no new valid assertion is derived, as follows:

[3]Note that these are intrascheme queries and that both the left-hand and the right-hand sides of the assertions above correspond to simple queries.

```
Procedure CP_Extraction (var CAD: a dictionary of assertions between complex
          patterns)
Var
   TCAD: a (temporary) dictionary of assertions between complex patterns;
Begin
   Repeat
      TCAD:= ;
      Derive_Intersection(CAD,TCAD);
      Derive_Union(CAD,TCAD);
      Derive_Forall(CAD,TCAD);
      Derive_Exists(CAD,TCAD);
      Derive_Complex(CAD,TCAD);
      Derive_Negation(CAD,TCAD);
      Discard_Weak(TCAD);
      Discard_Repeated(CAD,TCAD);
      CAD := CAD ∪ TCAD;
```

In the next subsections we will describe in details the procedures called within `CP_Extraction(CAD)`. Note that all of them derive properties of the form $<< L_1, L_2, f_{L1L2} >>$.

5.2.1. Computing object intersections and unions

The procedure *Derive_Intersection(CAD,TCAD)* derives assertions where L_1 is an intersection expression of two class expressions which are both subsets of a third one. More in details, suppose that the *CAD* stores $<< A, C, f_{AC} >>$ and $<< B, C, f_{BC} >>$; the procedure derives the inclusion coefficient of the assertion $<< A \sqcap B, C, f_{Expr} >>$. To this purpose, two extreme situations are considered: *(i)* *A* and *B* are included either ways into one another (best case), and *(ii)* *A* and *B* have minimal intersection (worst case). The general coefficient is then obtained as the mean value between those associated to the two extreme situations. The procedure is, therefore, as follows:

```
Procedure Derive_Intersection (CAD: a dictionary of assertions between complex
          patterns; var TCAD: a (temporary) dictionary of assertions between
          complex patterns)
Var
   $f_{Worst}$, $f_{Best}$ : Real ∈ [0,1]
Begin
   for each tuple << A, C, $f_{AC}$ >> ∈ CAD do
      for each tuple << B, C, $F_{BC}$ >>  ∈ CAD with A≠B do
            $f_{Worst}$ := max(0, $f_{AC}+f_{BC}-1$);
            $f_{Best}$ := min($f_{AC}$, $f_{BC}$);
            TCAD := TCAD ∪ << A ≈ B, C, ($f_{Worst}+f_{Best}$)/2 >>
         end
End
```

The procedure *Derive_Union(CAD,TCAD)* considers the case in which L_1 is obtained as the union of two class expressions. It is analogous to *Derive_Intersection(CAD,TCAD)* and, therefore, we will not illustrate its details.

5.2.2. Computing expressions containing $\forall$ and $\exists$

The procedure *Derive_Forall(CAD,TCAD)* derives expressions where the leftmost operator appearing in L_I is $\forall$. It considers all assertions of the form $<< \forall R[U].T_1{:}C_1, E, f_{C_1\forall E}$ belonging to the *CAD* and relative to a relationship *R* connected to the entity C_I through the role T_I and to the entity *E* through the role *U*. The algorithm works by case analysis, as follows:

```
Procedure Derive_Forall (CAD: a dictionary of assertions between complex
patterns;
          var TCAD: a (temporary) dictionary of assertions between complex
          patterns)
Begin
   for each tuple t = <<$\forall$R[U].$T_1$ :$C_1$, E, $f_{C1\forall E}$ >> $\in$ CAD do begin
      Compute_Subset(t,CAD,TCAD);
      Compute_Superset(t,CAD,TCAD);
      Compute_Further_Roles(t,CAD,TCAD)
   end
End
```

The procedure *Compute_Subset(t,CAD,TCAD)* verifies if a subset property between any expression *E'* and the entity C_I is known to hold; in the affirmative case, the procedure derives a new property and stores it in the *TCAD*.

In more details, suppose the assertion $<< E', C_1, f_{E'C_1} >>$ is stored in the *CAD*, where *E'* is any DL expression. From this, we infer $<< \forall R[U].T_1{:}E', E, f_{E'\forall E} >>$. For deriving $f_{E'\forall E}$ we observe that *(i)* say $f_{C_1\forall E}$ is the plausibility for an instance of *E* connected to *R* to be associated, through role T_I, only to instances of C_I; *(ii)* say $f_{E'\forall E}$ is the plausibility for an instance of *E* connected to the relationship *R* to be associated, through the role T_I, only to instances of *E'*; *(iii)* if μ is the average number of instances associated to the role T_I in *R*, then all μ instances of C_I must belong to *E'*[4]; the probability for this to happen is $(f_{E'C_1})^{\mu}$; therefore we can conclude that $f_{E'\forall E} = f_{C_1\forall E} \times (f_{E'C_1})^{\mu}$. The procedure is implemented as follows:

```
Procedure Compute_Subset (t: a Complex Pattern; CAD: a dictionary of assertions
          between complex patterns; var TCAD: a (temporary) dictionary of
          assertions between complex patterns)
Var
   $f_{E'\forall E}$ : Real;
   $\mu$: Integer;
Begin
   Let t = <<$\forall$R[U].$T_1$ :$C_1$, E, $f_{C1\forall E}$ >> belong to CAD;
   for each tuple t'=<<E',$C_1$,$f_{E'C1}$>> $\in$ CAD do begin
      Compute_Average_Fillers($T_1$,R,$\mu$);
      $f_{E'\forall E}$  = $f_{C1\forall E}$ $\times$ $(f_{E'C1})^{\mu}$ ;
      TCAD:=TCAD $\cup$ <<$\forall$R[U].$T_1$:E',E,$f_{E'\forall E}$>>
   end
End
```

[4] μ can be easily provided by database administrators.

The procedure *Compute_Average_Fillers(T_l,R,*m*)* considers the role T_l of the relationship *R* and determines the average number of fillers it has.

The procedure *Compute_Superset (t,CAD,TCAD)* verifies if an entity *F* exists including the entity *E*, according to assertions currently stored in the *CAD*; in the affirmative case, the procedure inserts a new assertion in the *TCAD*. The procedure is realized as follows:

```
Procedure Compute_Superset (t: a Complex Pattern; CAD: a dictionary of
assertions
          between complex patterns; var TCAD: a (temporary) dictionary of
          assertions between complex patterns)
Begin
   Let t=$<<\forall R[U].T_1 : C_1, E, f_{C1\forall E}>>$ belong to the CAD;
   for each tuple t'=$<<E,F,f_{EF}>> \in$ CAD do
      TCAD:=TCAD $\cup$ $<<\forall R[U].T_1:C_1,F,f_{C1\forall E} \times f_{EF}>>$
End
```

The procedure *Compute_Further_Roles(t,CAD,TCAD)* searches the CAD for the presence of an assertion of the form $<< \forall R[U].T_2{:}C_2, F, f_{C_2\forall E} >>$, where T_1 and T_2 are two different roles of the same relationship *R*. If one such assertion exists the procedure determines the inclusion coefficient associated to the assertion $<< \forall R[U].T_1{:}C_1, T_2{:}C_2, E, f_{Expr} >>$. Note that PDL expressions where two roles occur in the selection part are equivalent to intersection expressions. Therefore, f_{Expr} is computed in analogy to the intersection case. The procedure is as follows:

```
Procedure Compute_Further_Roles (t: a Complex Pattern; CAD: a dictionary of
          Assertions between complex patterns; var TCAD: a (temporary)
dictionary
          of assertions between complex patterns)
Var
   $f_{Worst}$, $f_{Best}$ : Real;
Begin
   Let t = $<<\forall R[U].T_1:C_1,E,f_{C1\forall E}>>$ belong to the CAD;
   for each tuple $t_1$ = $<<\forall R[U].T_2:C_2,E,f_{C2\forall E}>> \in$ CAD such that $T_1 \neq T_2$ do begin
      $f_{Worst}$ := max(0,$f_{C1\forall E}+f_{C2\forall E}-1$);
      $f_{Best}$ := min($f_{C1\forall E},f_{C2\forall E}$);
      TCAD:=TCAD $\cup$ $<<\forall R[U].T_1:C_1,T_2:C_2,E,(f_{Worst}+f_{Best})/2>>$
   end
End
```

This procedure is extended in the obvious way to derive expressions with any number of roles. The procedure *Derive_Exists(CAD,TCAD)* derives expressions whose leftmost operator is $ and behaves analogously to *Derive_Forall(CAD,TCAD)*.

5.2.3. Computing complex expressions

The procedure *Derive_Complex(CAD,TCAD)* is activated when two assertions of the form << *A*, *C*, f_{AC}>> and << *E*, *C*, f_{EC} >> are derived. If $f_{AC} < f_{EC}$ the plausibility coefficient f_{AE} to be associated with the assertion << *A*, *E*, f_{AE}>> can be evaluated. Again, two extreme situations must be considered:

(i) A and *E* have minimal intersection (worst case) and *(ii) A* and *E* are included into one another in either ways (best case). The procedure determines the plausibility coefficient of the worst and the best case, and takes the mean between the two values above as the value for f_{AE}:

```
Procedure Derive_Complex (CAD: a dictionary of assertions between complex
patterns;
          var TCAD: a (temporary) dictionary of assertions between complex
          patterns)
Var
   $f_{Worst}$, $f_{Best}$ : Real;
Begin
   for each pair of tuples << A,C,$f_{AC}$ >>,<<E,C,$f_{EC}$ >> ⊿ CAD such that $A \neq E$ do
      if $f_{AC} < f_{EC}$ then begin
         $f_{Worst}$ := max(0,$f_{AC}$+$f_{EC}$-1)/max($f_{AC}$,$f_{EC}$);
         $f_{Best}$ := min($f_{AC}$,$f_{EC}$)/max($f_{AC}$,$f_{EC}$);
         TCAD := TCAD ∪ <<A,E,($f_{Worst}$ + $f_{Best}$)/2 >>
      end
End
```

5.3. Computing Negation

The procedure *Derive_Negation(CAD,TCAD)* computes class negations. In Description Logics, the negation of a class represents all instances of the domain which are not instances of that class. In order to preserve safety, before the negation of a class is evaluated, the class itself must be intersected with one of its superset classes. Thus, deriving plausibility coefficient for a class negation is possible only if there exists an inclusion property relative to that class. The procedure for computing negation is as follows:

```
Procedure Derive_Negation (CAD: a dictionary of assertions between complex
          patterns; var TCAD: a (temporary) dictionary of assertions between
          complex patterns)
Begin
   for each tuple << B, A, $f_{BA}$ >> ∈ CAD do
      TCAD:=TCAD∪<<¬B,A,(1-$f_{BA}$) >>
End
```

It is worth pointing out that the negation of an object is itself an object. Therefore it can be exploited for deriving further complex properties.

5.4. Discarding weak or repeated patterns

The procedure *Discard_Weak(TCAD)* examines *TCAD* assertions and discards the weak ones, i.e. those having a plausibility coefficient under a given threshold.

The procedure *Discard_Repeated(CAD,TCAD)* checks, for each assertion in *TCAD*, if a corresponding assertion is already stored in *CAD*. In the affirmative case, the procedure discards that assertion having the weakest plausibility coefficient from the corresponding dictionary (*CAD* or *TCAD*).

5.5. Example

Consider, again, our example schemes reported in Figure 1 and in Figure 2. Suppose the following properties have been stored in the *IPD*:

$$<< \forall Works_{[AD]}[DL].IL{:}Engineer_{[AD]},\ Division_{[AD]},\ 0.4 >>$$

$$<< \forall Works_{[AD]}[DL].IL{:}Subordinate_{[PD]},\ Division_{[AD]},\ 0.5 >>$$

$$<< \forall Works_{[AD]}[DL].IL{:}(Engineer_{[AD]} \sqcap Employee_{[AD]}),\ Division_{[AD]},\ 0.4 >>$$

$$<< \forall Works_{[AD]}[DL].LC{:}(StrategicalCompetence_{[AD]} \sqcap Competence_{[AD]}),\ Division_{[AD]},\ 0.7 >>$$

then, the following assertions can be derived:

$$<< \forall Works_{[AD]}[DL].IL{:}(Engineer_{[AD]} \sqcap ItalianEngineer_{[AD]}),\ Division_{[AD]},\ 0.2916 >>$$

$$<< \forall Works_{[AD]}[DL].IL{:}(Engineer_{[AD]} \sqcap ItalianEngineer_{[AD]}),\ Department_{[PD]},\ 0.277 >>$$

$$<< \forall Works_{[AD]}[DL].LC{:}(StrategicalCompetence_{[AD]} \sqcap Competence_{[AD]}),\\ IL{:}(Engineer_{[AD]} \sqcap Employee_{[AD]}),\ Division_{[AD]}[,\ 0.25 >>$$

$$<< \forall Works_{[AD]}[DL].IL{:}(Engineer_{[AD]} \sqcap Employee_{[AD]}),\\ \forall Works_{[AD]}[DL].LC{:}(StrategicalCompetence_{[AD]} \sqcap Competence_{[AD]}),\ 0.357 >>$$

For instance, the last assertion says that the set of divisions in which all workers are engineers and employees is a subset of the set of divisions managing only strategic competencies, the plausibility coefficient being 0.357.

6. Applications of IKDD

PDL assertions describing intentional knowledge about sets of database schemes have many applications, and all of them can profitably exploit assertions between complex patterns as the ones we have presented in this paper. Applications include:

- the design of information integration layers on the top of existing database systems, such as mediators (Wiederhold, 1992; Ullman, 1997; Levy et al.,1996; Garcia-Molina et al., 1997) and, more in general, the development of tools for supporting the integrated access to Cooperative Information Systems (Papazoglou et al.,1992; Catarci and Lenzerini, 1993; Palopoli et al., 1998a,b,c);
- query optimization (Chaudhuri et al., 1995) and view materialization (Hanson, 1987);
- structuring and materialization of warehouses and constraints (Gupta et al., 1995).

In the following, we describe some examples of applications of complex pattern assertions to query optimization and view materialization.

6.1. Query Optimization

Complex assertions discovered by our IKDD algorithm can be exploited for the query optimization purposes. The key idea here is that queries can be expressed as Description Logic class expressions and that some relationships, in the form of PDL assertions, can be found to hold between the class expression denoting a certain query and other ones. These expressions can be exploited for solving some queries in a simpler, optimized way. There are also situations where the result of a query is predictable without executing it. In the following, some example cases are presented to illustrate these ideas more precisely.

6.1.1. Case 1

Assume we want to retrieve all the objects belonging to either the class A or to the class B or to the class C, i.e., we want to execute the query

$$Q = A \sqcup B \sqcup C$$

where classes *A*, *B* and *C* denote complex expressions. Suppose, moreover, that the assertion $<<B,C,W_{BC}>>$, with W_{BC} very high, has been derived and stored in the CAD. In this case, the query *Q* can be reduced to $A \sqcup C$

Example

Assume the following query must be executed:

$$Italian \sqcup Engineer \sqcup Employee$$

suppose, moreover, that the CAD stores

$$<<Engineer,\ Employee,\ 0.95>>$$

then, the previous query can be reduced to the following simpler form:

$$Italian \sqcup Employee$$

6.1.2. Case 2

This case is analogous to the previous one. Consider the following query

$$Q = A \sqcap B \sqcap C$$

where classes *A, B* and *C* denote complex expressions. Suppose that an assertion of the form $<<B,C,W_{BC}>>$ has been derived, where W_{BC} is very high. In this case the previous query can be reduced to $A \sqcap B$.

6.1.3. Case 3

The extraction of assertions may allow to remove redundant constraints. As an example suppose the following constraints have been defined for a database:

$$A \sqsubseteq B \qquad\qquad A \sqsubseteq C$$

Moreover, suppose that the CAD stores $<<B,\ C,\ W_{BC}>>$ where W_{BC} is high. Then the constraint $A \sqsubseteq C$ can be removed, since it is redundant.

Example

Assume that the following constraints have been stored for a query:

$Engineer \sqsubseteq Italian$ $\qquad\qquad$ $Engineer \sqsubseteq EuropeanUnionCitizen$

If the *CAD* stores:

$$<<Italian,\ EuropeanUnionCitizen,\ 1>>$$

the constraint $Engineer \sqsubseteq EuropeanUnionCitizen$ is redundant.

6.1.4. Case 4

Suppose the process of Intentional Knowledge Discovery extracted the following assertions:

$$<< \exists R[U].T_1{:}C_1, A, W_{Expr1} >> \qquad\qquad << \exists R[U].T_2{:}C_2, B, W_{Expr2} >>$$

Suppose the following query must be executed:

$$Q = \exists R[U].T_1{:}C_1, T_2{:}C_2$$

and suppose that also the following assertions have been previously derived:

$$<<A,\ C,\ W_{AC}>> \qquad\qquad << B, \neg C, W_{B\neg C} >>$$

denoting that *A* is a subset of *C* with plausibility W_{AC} and that *B* is a subset of $\neg C$ with plausibility $W_{B\neg C}$. If W_{AC} and $W_{B\neg C}$ are high, we can conclude that the result of *Q* is empty, without any need to actually retrieve data.

Example

Suppose the following assertions belong to the CAD:

$$<<\exists \mathit{Works[DL].IL{:}Engineer},\ \mathit{ManagementDivision},\ 0.7>>$$

$$<<\exists \mathit{Works[DL].IS{:}LowSalary},\ \mathit{SecondaryDivision},\ 0.8>>$$

the first one indicates that divisions in which at least an engineer works is a subset of management divisions with plausibility 0.7; the latter one denotes that divisions in which at least one person earns a low salary form a subset of Secondary Divisions with plausibility 0.8.

Suppose that the CAD also stores:

$$<<\mathit{ManagementDivision},\ \mathit{StrategicalDivision},\ 0.98>>$$

$$<<\mathit{SecondaryDivision},\ \neg \mathit{StrategicalDivision},\ 0.95>>$$

Finally, assume that the following query is to be executed:

$$Q = \exists \mathit{Works[DL].IL{:}Engineer},\ \mathit{IS{:}\ LowSalary},$$

i.e., we look for divisions where at least an engineer earning a low salary works. Then, we can conclude, with high plausibility, that the result of the query is the empty set without actually executing it.

6.1.5. Case 5

Suppose that the following complex assertion has been derived:

$$<<\ \exists R[U].T_l{:}C_l,\ \perp,\ W_{Expr}\ >>.$$

and assume the following query must be executed:

$$Q = \forall R[U].T_l{:}C_l$$

By noting that, clearly, the following assertion is valid:

$$<<\forall R[U].T_l{:}C_l,\ \exists R[U].T_l{:}C_l,\ 1\ >>$$

if W_{Expr} is very high, we can conclude that the empty set is the query answer. This result also holds for any query Q such that an assertion of the form $<<Q,\ \exists R[U].T_l{:}C_l, W_Q>>$ has been derived and W_Q is high.

Example

Suppose that the assertion

$$<<\exists \mathit{Works[DL].IL{:}Engineer},\ \perp,\ 0.95\ >>$$

has been obtained, indicating that there is no division where at least one engineer works.

If the query

$$\forall \mathit{Works[DL].IL{:}Engineer}$$

is to be executed asking for divisions in which the personnel consists of engineers only, the empty set can be immediately yielded as the query result.

6.2. View Materialization

The availability of assertions about complex object patterns can be used to decide a criterion guiding view materialization. For deriving a view materialization criterion the following reasoning can be drawn: suppose a complex query *B*, involving a great number of objects, must be executed; the result of its execution can be considered as a virtual view. Suppose, now, that another complex query *A* must be executed. As usual *A* and *B* can be looked at as PDL class expressions. Suppose that the assertion $<<A, B, W_{AB}>>$ has been derived applying IKDD techniques and that W_{AB} is large. Then, the view *B* can be conveniently materialized and the query *A* can be efficiently executed on the materialized view corresponding to *B*.

Obviously we cannot materialize all possible views, therefore the necessity arises of a criterion for determining which views are to be materialized. The problem is difficult because it is hard to figure out which queries are going to be executed in the future. However, we argue that the greater the number of inclusion properties in which a given expression *E* appears as a superset is, the higher the probability of answering queries exploiting the view corresponding to *E* is. For a fixed given number of inclusion properties in which *E* appears, the greater the values of inclusion coefficients associated to *E* are, the most convenient the materialization of views corresponding to *E* is.

The algorithm for view materialization is, therefore, the following:

```
Algorithm for view materialization
Input:  a dictionary CAD of assertions between complex patterns;
Output: a set MV of views to be materialized;
Const
   $D_M$ := 1.5;
Var
   TMVD: a (temporary) materialized view dictionary;
   Presence_Number: Integer;
   Sum, Interest_Coefficient, $V_{Max}$, $V_{Min}$: Real;
Begin
   MV := □;
   TMVD := □;
   for each tuple <<$CE_1$, $CE_2$, $f_{12}$>> ∈ CAD do
      if not Belong($CE_2$, TMVD) then begin
         Presence_Number:=0;
         Sum:=0;
         for each tuple <<$CE_i$, $CE_2$, $f_{ij}$>> do begin
            Inc(Presence_Number);
            Sum := Sum + $f_{ij}$
         end
         Interest_Coefficient := $\beta$ × Presence_Number + Sum;
         TMVD := TMVD ∪ ($CE_2$, Interest_Coefficient)
      end;
   Compute_Max(TMVD,$V_{Max}$);
   Compute_Min(TMVD,$V_{Min}$);
   $Th_M$ := ($V_{Max}$ + $V_{Min}$)/$D_M$;
   for each tuple ($CE_i$,$IC_i$) ∈ TMVD such that $IC_i \geq th_M$ do
      MV := MV ∪ $CE_i$
End
```

Here D_M is a constant used for tuning up the threshold of interest: the higher D_M is, the lower the threshold will be. *TMVD* is a temporary support dictionary storing tuples of the form (T_i, C_i), where T_i is a view and C_i is an associated interest coefficient. The function Belong(E,TMVD) yields TRUE if onetuple of the form *(E,C)* belongs to *TMVD*, FALSE otherwise. The procedure Compute_Max(TMVD,V_{Max}) (Compute_Min(TMVD,V_{Min})) computes the maximum (minimum) coefficient in *TMVD* and stores it in V_{Max} (V_{Min}).

7. Conclusions

In this paper we have illustrated techniques for deriving complex assertions relating objects belonging to database schemes. Our techniques are explicitly designed to work on large numbers of input scheme objects.

The main idea underlying our approach is that of adapting the principle of Knowledge Discovery in Databases to extracting knowledge from database scheme catalogues. We have called the resulting approach Intentional Knowledge Discovery (IKDD).

IKDD uses a fuzzy variant of Description Logic, called PDL, for representing and reasoning about scheme properties. We have illustrated also the application of derived assertions to query optimization and view materialization problems.

The methods we have presented here have been implemented as part of a more general design support tool, called D.I.K.E. (Database Intentional Knowledge Extractor) we are constructing at University of Calabria.

References

Abiteboul, S., 1997,"Querying Semi-Structured Data", *Proc. ICDT '97,* 1-18, Delphi, Greece.

Batini, C., Lenzerini, M., 1984, "A methodology for data schema integration in the entity relationship," model, *IEEE TSE* 10(6), 650-664.

Castano, S., De Antonellis, V., 1997, "Semantic Dictionary Design for Database Interoperability," *Proc. of ICDE'97*, Birmingham, United Kingdom.

Catarci, T., Lenzerini, M., 1993, "Representing and using interschema knowledge in cooperative information systems," *Journal of Intelligent and Cooperative Information Systems*, 2(4), 375-398.

Chaudhuri, S., Krishnamurthy, R., Potamianos,S., Shim,K., 1995, "Optimizing queries with materialized views," *Proceedings of ICDE '95*, 190-200, Taipei, Taiwan.

Fankhauser, P., Kracker, M., Neuhold,E.J., 1991," Semantic vs. Structural Resemblance of Classes," *SIGMOD RECORD*, 20(4), 59-63.

Fayyad, U.,Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., 1996, *Advances in Knowledge Discovery and Data Mining*, The AAAI - The MIT press.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J.,Vassalos, V.,Widom, J., 1997, "The TSIMMIS Approach to Mediation: Data Models and Languages," *Journal of Intelligent Information Systems* 8, 117-132.

Gupta, A., Mumick, I.S., Ross, K.A., 1995, "Adapting materialized views after redefinitions," *Proc. ACM SIGMOD.*

Hanson, E.H., 1987, "A Performance Analysis of View Materialization Strategies," Proc. *SIGMOD '87,* 440-453, San Francisco (California), USA.

Levy, A., Rajaraman, A., Ordille, J., 1996, "Querying heterogeneous information sources using source descriptions," *Proc. VLDB '96*, 251-262, Bombay, India.

Palopoli, L., Saccà, D., Ursino, D., 1998a ,"Semi-automatic, semantic discovery of properties from database schemes," *Proc. IDEAS'98*, 244-253, IEEE Press, Cardiff, United Kingdom.

Palopoli, L., Saccà, D.,Ursino, D., 1998b,"Automatic Derivation of Terminological Properties from Database Schemes," *Proc. DEXA '98*, LNCS, Springer Verlag, 90-99, Wien, Austria.

Palopoli,L., Saccà, D., Ursino, D., 1998c, "An Automatic Technique for Detecting Type Conflicts in Database Schemes," *Proc. ACM CIKM'98*, 306-313, Bethesda (Maryland), USA.

Papazoglou, M.P., Laufmann, S.C., Sellis, T.K., 1992, "An organizational framework for cooperative information systems,"*Journal of Intelligent and Cooperative Information Systems*, 1(1).

Spaccapietra, S., Parent, C., 1994, "View Integration: A Step Forward in Solving Structural Conflicts," *IEEE TKDE* 6(2), 258-274.

Ullman, J.D., 1997, "Information integration using logical views," *Proc. ICDT '97*, Delphi, Greece, 19-40.

Wiederhold, G., 1992, "Mediators in the architecture of future information systems,"*IEEE Computer*, 25, 38-49.