Describing Database Objects in a Concept Language Environment

Alessandro Artale, Francesca Cesarini, and Giovanni Soda

Abstract—In this paper, we formally investigate the structural similarities and differences existing between object database models and concept languages establishing a correspondence between the two environments. Object Databases Models deal with two kinds of data: individual objects, which have an identity, and values, which can be basic values or can have complex structures containing both basic values and objects. Concept Languages only deal with individual objects. The correspondence points out the different role played by objects and values in both approaches and defines a way of properly mapping database descriptions into concept language descriptions at both a terminological and assertional level. Once the mapping is achieved, object databases can take advantage of both the algorithms and the results concerning their complexity developed in concept languages.

Index Terms—Concept languages, object databases, knowledge representation.

1 INTRODUCTION

CONCEPT *Languages* [12], [18], [20] are developed in the Knowledge Representation research area for representing object class knowledge. They describe the structure of objects at a terminological level by means of concepts[1] (one place predicates) and roles (two place predicates), and an external denotational semantics gives meaning to the terms used in the descriptions. Concept languages can also be used to make assertions about individual objects, i.e., to state that an object is in the extension of a concept and that a pair of objects is in the extension of a role. Further, concepts can be distinguished as *primitive concepts*, where the concept structure is interpreted as a set of necessary conditions and *defined concepts*, where the concept structure is interpreted as a *definition*, i.e., a set of necessary and sufficient conditions that must be satisfied by their instances. Thus the *extension* of a defined concept corresponds to the domain object set whose structure conforms to that description. Deductive reasoning at both a terminological and assertional level is being widely studied: *subsumption* computation determines whether a subset relationship exists between two concept denotations. The subsumption computation provides a reasoning capability that can be used for investigating both the structural characteristics of classes and the relationships between them and specific objects; these subsumption based inferences constitute what is called taxonomic reasoning. Concept languages exploit taxonomic reasoning in a number of applications: it allows to classify concepts into a taxonomic graph with subsumption as partial order relation, to verify both the consistency of a set of class descriptions and the consistency of instance assertions with respect to their class definitions, or to find the most appropriate class an object can belong to and to optimize query answering. Systems like BACK [19], CANDIDE [5], CLASSIC [9], KRIS [4], KRIPTON [8], LOOM [17], NIKL [21], YAK [11] describe a world of objects and relationships among objects, and directly exploit the above mentioned features.

The structural aspects of *Object database models* [1], [15] traditionally refer to *tuple-constructor-based* class descriptions and **isa** hierarchies. They are concerned with a world of individual objects and values and their mutual relationships; values are explicitly dealt with because object descriptions very often use complex values that are local to them [15]. The tuple constructor, specifying relationships between instances of different types, captures an expressive power similar to that of attaching roles to concept descriptions; furthermore, this constructor allows to describe both objects and values. An **isa** expresses a subset relationship between classes; it is related to the subtyping notion (i.e., to syntactic constraints on class descriptions) but it must be explicitly stated: A **isa** relationship cannot be inferred from the class structure. For this reason, object databases only capture the semantics of primitive classes, which description indicates only necessary conditions for an object to belong to this class.

Recently, inference techniques derived from concept languages have been applied to object database models [3], [6]. A common aspect of these studies is that they consider object database models enriched by the notion of defined classes. This allows to adapt taxonomic reasoning to deal with the peculiarity of object database model environments and therefore obtain a database equipped with inference capabilities. Exploiting taxonomic reasonig in object database models can be profitable for many database topics on both intensional and extensional levels. As illustrated in [2], [3], [6], [7], **isa** relationships which can be inferred from class descriptions are made explicit; in other words, the user's taxonomy is enriched with implicit **isa** relationships; the user **isa** relationships are checked with respect to the subsumption relations; the schema consistency is checked by discovering cycles and incoherent classes (i.e., classes with always empty extension); the schema can be transformed into a *minimal form* where the redundancies with respect to inheritance are removed; query evaluation can be optimized finding the correct placement of a query object in a given taxonomy; individual objects are *recognized* to belong to a class abstracting their properties and classifying the resulting abstraction (this inference is called *instance recognition*). Taxonomic reasoning only refers to object structural characteristics, and many other aspects, such as, methods, constraints, etc., are not dealt with. Nevertheless, it is able to provide for a powerful reasoning capability even if it only focuses on some object characteristics.

Since the integration of object and concept environments is being fruitful, our study aims at formally investigating relationships between concept languages and object database models, in order to point out the similarities and differences existing between the two environments. Our work originates from the above mentioned studies, thus we only focus on the structural characteristics of objects. Our view is a model-theoretic one, we illustrate how object database descriptions can be transformed into concept language descriptions by a suitable mapping, capable of maintaining satisfiability. This kind of mapping provides a common framework for evaluating and comparing different object database models with respect to the corresponding concept languages, and can be also exploited for analyzing the nature of the features supported by both objects databases and concept languages. Moreover, this mapping allows us to inherit results deriving from concept languages (which have been thoroughly studied), such as complexity results and algorithm techniques.

The object database model and the concept language we refer to are described in Sections 2 and 3. In Section 4, we define a way of mapping database schema descriptions into concept language descriptions while maintaining satisfiability, and an assertional mapping that maintains consistency is discussed in Section 5. Section 6 contains our concluding remarks.

1. In the following, we will use the terms *concept* and *class* interchangeably.

• *The authors are with the Dipartimento di Sistemi ed Informatica, Università di Firenze, via di Santa Marta, 3, 50139 Firenze, Italy.*
E-mail: {alex, giovanni}@ingfi1.ing.unifi.it.

Manuscript received Feb. 16, 1993; revised Mar. 30, 1994.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number K95089.

2 The Object Database Model

We consider an Object Database Model that supports the main structural features usually present in this field (see, for example [1], [10], [15]). The main structure of our model is the *Class* that denotes sets of *Objects*, each of which is identified by an *Object Identifier*. Objects in classes can have a complex structure obtained by repeatedly using the *tuple* and *set* constructors; therefore, the *type system* is based on the most widely used type constructors. *Type names* are provided for simplifying user declarations. The set type allows us to distinguish between single and multivalued attributes; furthermore, we consider set types with cardinality constraints, integrating in the schema description a kind of integrity constraint in the database environment. As regards classes, we distinguish between *primitive* and *defined* classes; introducing defined classes allows us to use taxonomic reasoning for database objects [3].

<declaration> := <type-declaration> | <class-declaration>
<type-declaration> := **type** <type-id> = <tuple-type>
<class-declaration> := ⊤ | **class** <class-id> <prim-def> **isa** <class-id>* <tuple-type>
<type> := <tuple-type> | <class-id> | <type-id> | <basic-type>
<tuple-type> := [<component>*]
<component> := <label>:<type> | <label>:<set-type>
<set-type> := {<type>}$_{min,\, max}$
<basic-type> := string | integer | b_i
<prim-def> := $\dot{\leq}$ | $\doteq$

Fig. 1. Database syntax.

The database schema S can be defined by means of the syntax in Fig. 1. Recursive definitions are not allowed. Basic types can include other types besides string and integer; anyway, basic types indicate countable and nonfinite sets. This syntax allows, for instance, to describe the concept of father as the set of all "individuals who are persons with at least one child and all children are persons":

$$\textbf{class } \mathit{Father} \doteq \textbf{isa } \mathit{Person}[\mathit{child}: \{\mathit{Person}\}_{1,\infty}]$$

Given a set of declarations, i.e., a database schema S, an *interpretation* $\mathcal{I}_{\mathcal{D}} = (\mathcal{V}, \delta, \cdot^{\mathcal{I}_{\mathcal{D}}})$ consists of:

- A set $\mathcal{V}$ of values (the *domain* of $\mathcal{I}_{\mathcal{D}}$), $\mathcal{V} = \mathcal{B} \cup O \cup \mathcal{V}_{\mathcal{T}} \cup \mathcal{V}_S$, with:
  1) $\mathcal{B} = \bigcup_{i=1}^{n} \mathcal{B}_i$, $\mathcal{B}_i$ set of values associated with each basic type; $\mathcal{B}_i \cap \mathcal{B}_j = \emptyset, \forall i, j, i \neq j$.
  2) O is a countable set of symbols called object identifiers disjoint from $\mathcal{B}$.
  3) $\mathcal{V}_{\mathcal{T}}$ is the set of *tuple values*: $\mathcal{V}_{\mathcal{T}} = \{v_t \mid v_t$ *is a mapping from the set of labels to* $\mathcal{V}\}$. We denote by $[l_1 : v_1 \ldots l_k : v_k]$ the mapping defined on $\{l_1 \ldots l_k\}$ such that $v_t(l_i) = v_i \in \mathcal{V}$, i = 1, ..., k.
  4) $\mathcal{V}_S$ is the set of *set-values*: $\mathcal{V}_S = \{v_s \mid v_s \subseteq \mathcal{V}\}$. A set-value is denoted by $\{v_1, \ldots, v_k\}$ such that $v_i \in \mathcal{V}$, i = 1, ..., k.
- A mapping δ that associates a tuple value with each object-identifier: $\delta : O \rightarrow \mathcal{V}_{\mathcal{T}}$
- An interpretation function $\cdot^{\mathcal{I}_{\mathcal{D}}}$ that maps every syntactic constructor to a subset of $\mathcal{V}$ in such a way that:

$$(<\text{basic-type}>_i)^{\mathcal{I}_{\mathcal{D}}} = \mathcal{B}_i$$

$$(<\text{tuple-type}>)^{\mathcal{I}_{\mathcal{D}}} = ([l_1 : t_1 \ldots l_k : t_k])^{\mathcal{I}_{\mathcal{D}}} =$$
$$\{v_t \in \mathcal{V}_{\mathcal{T}} \mid v_t \text{ is defined atleast on } \{l_1 \ldots l_k\} \text{ and } v_t(l_i) \in (t_i)^{\mathcal{I}_{\mathcal{D}}},\ i = 1, \ldots, k\}$$

$$(<\text{set-type}>)^{\mathcal{I}_{\mathcal{D}}} = (\{t\}_{m,n})^{\mathcal{I}_{\mathcal{D}}} = \{v_s \in \mathcal{V}_S \mid m \leq \left\| v_s \subseteq t^{\mathcal{I}_{\mathcal{D}}} \right\| \leq n\}$$

$$(<\text{type-id}>)^{\mathcal{I}_{\mathcal{D}}} = (\textbf{type}\ \text{type-id} = <\text{tuple-type}>)^{\mathcal{I}_{\mathcal{D}}}$$
$$= (<\text{tuple-type}>)^{\mathcal{I}_{\mathcal{D}}}$$

$$(<\text{class-id}>)^{\mathcal{I}_{\mathcal{D}}} =$$
$$\begin{cases} (\top)^{\mathcal{I}_{\mathcal{D}}} = O & \text{Universal Class} \\ (\textbf{class } C_d \doteq \textbf{isa } C_1 \ldots C_n <\text{tuple-type}>)^{\mathcal{I}_{\mathcal{D}}} = & \\ \bigcap_{i=1}^{n} (C_i)^{\mathcal{I}_{\mathcal{D}}} \bigcap \{o \in O \mid \delta(o) \in (\mathrm{TYP}(\mathrm{C_d}))^{\mathcal{I}_{\mathcal{D}}}\} & \text{Defined Class} \\ (\textbf{class } C_p \dot{\leq} \textbf{isa } C_1 \ldots C_n <\text{tuple-type}>)^{\mathcal{I}_{\mathcal{D}}} \subseteq & \\ \bigcap_{i=1}^{n} (C_i)^{\mathcal{I}_{\mathcal{D}}} \bigcap \{o \in O \mid \delta(o) \in (\mathrm{TYP}(C_p))^{\mathcal{I}_{\mathcal{D}}}\} & \text{Primitive Class} \end{cases}$$

For the interpretation of TYP(C) we have the following recursive definition.

Definition 1 (Interpretation of TYP(C)): *Given the class declarations:*

class C_0 <prim-def> <tuple-type>$_0$

class C_1 <prim-def> **isa** C_0 <tuple-type>$_1$

....

class C_{n+1} <prim-def> **isa** $C_0 \ldots C_n$ <tuple-type>$_{n+1}$

we have

$$(\mathrm{TYP}(C_0))^{\mathcal{I}_{\mathcal{D}}} = (<\text{tuple-type}>_0)^{\mathcal{I}_{\mathcal{D}}}$$
$$(\mathrm{TYP}(C_1))^{\mathcal{I}_{\mathcal{D}}} = (\mathrm{TYP}(C_0))^{\mathcal{I}_{\mathcal{D}}} \cap (<\text{tuple-type}>_1)^{\mathcal{I}_{\mathcal{D}}}$$
$$\ldots\ldots \quad \ldots\ldots \quad \ldots\ldots$$
$$(\mathrm{TYP}(C_{n+1}))^{\mathcal{I}_{\mathcal{D}}} = \bigcap_{i=0}^{n} (\mathrm{TYP}(C_i))^{\mathcal{I}_{\mathcal{D}}} \cap (<\text{tuple-type}>_{n+1})^{\mathcal{I}_{\mathcal{D}}}$$

This semantics, allows us to consider the **isa** relationship as an inclusion between classes. Moreover, each value can have more than one type [10]: When a value is of type t, then it is of type t', too, in the case that $(t)^{\mathcal{I}_{\mathcal{D}}} \subseteq (t')^{\mathcal{I}_{\mathcal{D}}}$. Remark that the interpretation of a class is a set of objects which have a value according to the type of the class; furthermore, these objects must also belong to the interpretation of the classes appearing in the **isa** clause.

As regards the notion of primitive/defined class, the interpretation of a defined class consists of all the objects verifying the above-mentioned constraints, while the interpretation of a primitive class is a subset of them. For example, in the case of

$$\textbf{class } \mathit{Person} \dot{\leq} [\mathit{name}:\mathit{string}\ \mathit{birthdate}:\mathit{Date}]$$
$$\textbf{class } \mathit{Project} \doteq [\mathit{proj-code}:\mathit{string}\ \mathit{description}:\mathit{string}]$$
$$\textbf{class } \mathit{Student} \doteq \textbf{isa } \mathit{Person}[\mathit{regist-num}:\mathit{int}\ \mathit{enrolled}:$$
$$\mathit{College}\ \mathit{enrolled-course}:\mathit{string}]$$

the interpretation of *Person* is a subset of the objects having a *name* and a *birthdate*, while the interpretation of *Project* is the set of all the objects having a *proj-code* and a *description*. Thus, an object having a *name* and a *birthdate* must be explicitly asserted belonging to the *Person* class, while an object having a *proj-code* and a *description* always belongs to *Project*. The interpretation of *Student* consists of all the objects that belong to *Person*, have a *regist-num*, are *enrolled* to a *College*, and are *enrolled* to some *courses*. Because of the recursive definition of the TYP's interpretation, these objects also have a *name* and a *birthdate* (inherited from *Person*). As a matter of fact, the tuple values of the *Student* objects are obtained by intersecting tuple values defined at least on the *name* and *birthdate* labels, and tuple values defined at least on the *regist-num*, *enrolled* and *enrolled-course* labels. If we know that an object belongs to class *Person* and its value is defined at least on labels *name*, *birthdate*, *regist-num*, *enrolled*, and *enrolled-course*, we can deduce that this object belongs to *Student*, too. In general, fixed the interpretation of the primitive classes, the interpretation of the defined classes is unambigously determined.

An interpretation $\mathcal{I}_{\mathcal{D}}$ is a *model* for a class C if $C^{\mathcal{I}_{\mathcal{D}}} \neq \emptyset$. If a class has a model, then it is *satisfiable*; otherwise it is *unsatisfiable*. A class C is *subsumed* by a class D (written $C \sqsubseteq D$) if $C^{\mathcal{I}_{\mathcal{D}}} \subseteq D^{\mathcal{I}_{\mathcal{D}}}$ for every interpretation $\mathcal{I}_{\mathcal{D}}$. The satisfiability notion can be extended to generic syntactic constructors. Let t be a syntactic constructor, then t is *satisfiable* if there exists an interpretation $\mathcal{I}_{\mathcal{D}}$ such that $t^{\mathcal{I}_{\mathcal{D}}} \neq \emptyset$.

In our framework, every **isa** clause corresponds to a subsumption relationship: if C_1**isa**C_2 then C_2 subsumes C_1. The opposite is not necessarily true; a class can subsume another one even if subsumption is not explicitly defined by means of an **isa** clause. Because our interpretation function is totally based on structural characteristics, the meaning of a structured description is only determined by its internal structure. This allows us to make an algorithm to deduce all the subsumption relationships among classes implicitly given by the structural conditions appearing in the class descriptions. The algorithm that computes subsumption between classes is sound and complete, and is polynomial in the size of a class [3].

3 The Concept Language

We strictly follow the concept language formalism introduced by [20] and further elaborated by [4], [12], [14], among others. We examine the minimal concept language that covers our object database model; *concept terms* (denoted by the letters C and D) are built out of *atomic concepts* (denoted by the letter *A*), *roles* (denoted by the letter *R*) and *features* (denoted by the letter *f*) according to the following syntax rule:

$$C, D \rightarrow A \;|\top|\; C \sqcap D \;|\forall_{[m,n]}R.C|\; f{:}C$$

An *interpretation* $\mathcal{I}_C = (\Delta^{\mathcal{I}_C}, \cdot^{\mathcal{I}_C})$ consists of a set $\Delta^{\mathcal{I}_C}$ (the *domain* of $\mathcal{I}_C$) and a function $\cdot^{\mathcal{I}_C}$ (the *interpretation function* of $\mathcal{I}_C$) that maps every concept term to a subset of $\Delta^{\mathcal{I}_C}$, every role to a subset of $\Delta^{\mathcal{I}_C} \times \Delta^{\mathcal{I}_C}$ and every feature to a partial function $f^{\mathcal{I}_C}$ from $\Delta^{\mathcal{I}_C}$ to $\Delta^{\mathcal{I}_C}$ (we denote the domain of $f^{\mathcal{I}_C}$ as $domf^{\mathcal{I}_C}$) in such a way that the following equations are satisfied:

$$\begin{aligned}
\top^{\mathcal{I}_C} &= \Delta^{\mathcal{I}_C} \\
(C \sqcap D)^{\mathcal{I}_C} &= C^{\mathcal{I}_C} \cap D^{\mathcal{I}_C} \\
(f{:}C)^{\mathcal{I}_C} &= \{a \in domf^{\mathcal{I}_C} \mid f^{\mathcal{I}_C}(a) \in C^{\mathcal{I}_C}\} \\
(\forall_{[m,n]}R.C)^{\mathcal{I}_C} &= \{a \in \Delta^{\mathcal{I}_C} \mid m \leq \|\{b \in \Delta^{\mathcal{I}_C} \mid (a,b) \in R^{\mathcal{I}_C}\| \\
&\leq m \wedge \forall b.(a,b) \in R^{\mathcal{I}_C} \rightarrow b \in C^{\mathcal{I}_C}\}
\end{aligned}$$

Features were recently introduced for distinguish between arbitrary binary relations (roles) and functions (features) [4], [14], [18]. The presence of roles and features allows us to distinguish between single-valued and multivalued attributes in investigating the correspondence between object database models and concept languages.

An interpretation $\mathcal{I}_C$ is a *model* for a concept term C if $C^{\mathcal{I}_C} \neq \emptyset$. If a concept term has a model, then it is *satisfiable*, otherwise it is *unsatisfiable*. A concept term C is *subsumed* by a concept term D (written $C \sqsubseteq D$) if $C^{\mathcal{I}_C} \subseteq D^{\mathcal{I}_C}$ for every interpretation $\mathcal{I}_C$. Subsumption can be reduced to satisfiability since C is subsumed by D if and only if $C \sqcap \neg D$ is not satisfied.[2]

Let *A* be an atomic concept and C be a concept term, one can introduce descriptions for atomic concepts by *terminological axioms* of the form $A \dot{\leq} C$ and $A \doteq C$. An interpretation $\mathcal{I}_C$ satisfies $A \dot{\leq} C$ iff $A^{\mathcal{I}_C} \subseteq C^{\mathcal{I}_C}$, while it satisfies $A \doteq C$ iff $A^{\mathcal{I}_C} = C^{\mathcal{I}_C}$. Furthermore, we denote as an *undescribed concept* an atomic concept that never appears as the first argument of a terminological axiom. A *terminology* $\mathcal{T}$ is a finite set of terminological axioms with the additional restriction that

1) every atomic concept may appear only once as the first argument of a terminological axiom in $\mathcal{T}$, and
2) $\mathcal{T}$ must not contain cyclic definitions.

Let *Person* be a concept, *child* be a role, and *wife* be a feature. The following axioms

$$\begin{aligned}
Female &\dot{\leq} Person \\
HappyFather &\doteq Person \sqcap \forall_{1,3}child.Person \sqcap wife{:}Person \\
VeryHappyFather &\doteq Person \sqcap \forall_{1,3}child.Female \sqcap wife{:}Person
\end{aligned}$$

express that:

1) A Female is a Person;
2) A *HappyFather* is exactly a *Person* with at least one and at most three *children*, that are *Persons*, and a *wife* that is a *Person*;
3) A *VeryHappyFather* is exactly a *Person* with at least one and at most three *chidren* that are *Females*, and a *wife* that is a *Person*.

Furthermore, the concept *HappyFather* subsumes *VeryHappyFather*.

4 Mapping

At this point we show how it is possible to translate a database schema into a concept language description while maintaining satisfiability during mapping. Each class declaration is translated into a terminological axiom by mapping the **isa** clause to a conjunction of concepts, while each tuple type gives rise to a conjunction of feature or role restrictions, according to whether or not the label in the tuple type is single- or multivalued. Furthermore, atomic disjoint concepts are introduced in order to preserve the disjointness between classes, basic types and tuple types.

DEFINITION 2 (Syntactic Mapping): *Let $\mathcal{N}$ be a function from database class declarations to terminological axioms. Given the database class declaration:* **class** C <prim–def> **isa** $C_1 \ldots C_n$ $[l_1 : t_1 \ldots l_k : t_k]$; *then* $\mathcal{N}$[<class–declaration>] *is the terminological axiom:*

$$A < \text{prim-def} > A_C \sqcap \mathcal{N}[C_1] \sqcap \ldots \sqcap \mathcal{N}[C_n] \sqcap \mathcal{N}[l_1{:}t_1] \sqcap \ldots \sqcap \mathcal{N}[l_k{:}t_k]$$

where the following equations hold:

- $\mathcal{N}[l_i{:}t_i] = f_{l_i}{:}\mathcal{N}[t_i]$; *if t_i is not a set type*
- $\mathcal{N}[l_i{:}t_i] = \forall_{[m,n]}R_{l_i}.\mathcal{N}[t_i']$; if $t_i = \{t_i'\}_{m,n}$
- *For $\mathcal{N}[t_i]$ we have:*
  1) *Let $t_i = b_i$ a basic type, then:* $\mathcal{N}[t_i] = \mathcal{N}[b_i] = A_{b_i}$
  2) *Let $t_i = C_i$ a class name, then:* $\mathcal{N}[t_i] = \mathcal{N}[C_i] = A_i$ *and if A_i is an undescribed concept then* $A_i \dot{\leq} A_C$
  3) *Let* $t_i = [l_{1_i}{:}t_{1_i} \ldots l_{k_i}{:}t_{k_i}]$, *then:*

2. Although $C \sqcap \neg D$ does not belong to the original language, we can use a modified set of rules, borrowed from a more expressive language with existential quantification and negation of primitive concepts. This leads to an algorithm for satisfiability (see [13] for more details).

$$\mathcal{N}[t_i] = A_{\mathcal{V}_T} \sqcap \mathcal{N}[l_{1_i}:t_{1_i}] \sqcap \ldots \sqcap \mathcal{N}[l_{k_i}:t_{k_i}]$$

4) *Let $t_i = T_i$ be a type name described as:*

$$\textbf{type } T_i = [l_{1_i}:t_{1_i} \ldots l_{k_i}:t_{k_i}],$$

then $\mathcal{N}[T_i] = \mathcal{N}[l_{1_i}:t_{1_i} \ldots l_{k_i}:t_{k_i}]$

- $A_C, A_{b_i}, A_{\mathcal{V}_T}$ *are*[3] *undescribed concepts pairwise disjoint.*

As an example of the previously defined mapping, we examine the following class declaration:

$$\textbf{class } College \doteq [name:string \;\; has-courses:\{Course\}_{1,\infty} \;\; address:[street:string \;\; city:string]]$$

its translation via $\mathcal{N}$ is therefore

$$College \doteq A_C \sqcap name:String \sqcap \forall_{1,\infty} has-courses.Course \sqcap address:(A_{\mathcal{V}_T} \sqcap street:String \sqcap city:String)$$

It is interesting to note that a mapping that maintains an isomorphism between these two environments has to introduce some new primitive classes (A_C, $A_{\mathcal{V}_T}$) in such a way that classes are disjoint from tuple types. As a matter of fact, classes are collections of objects which are semantically different from tuple values, being $\mathcal{O}$ and $\mathcal{V}_T$ disjoint sets. For example, given the class descriptions

$$\textbf{class } Organization \doteq [name:string \;\; city:string]$$
$$\textbf{class } Employee1 \doteq \textbf{isa } Person[works-at:Organization]$$
$$\textbf{class } Employee2 \doteq \textbf{isa } Person[works-at:[name:string \;\; city:string]]$$

the interpretation of *Employee1* is different from the interpretation of *Employee2*. In the former case, a class *Employee1* object is related, by means of the *works-at* label, to a class *Organization* object, while in the latter, a class *Employee2* object is related to a locally defined tuple value. Then, we do not associate to a locally defined value an object identifier nor create a new membership class [15]. Given that in the database environment a great effort is devoted to optimize the amount of information to be stored, the possibility of managing local information allows us to respect the above cited principle. On the contrary, concept languages do not make this distinction, dealing only with individuals. Then, the only way to mantain disjoint the set of fillers for the *works-at* label after the mapping of *Employee1* and *Employee2*, is to introduce the class A_C when mapping the database class *Organization* and the class $A_{\mathcal{V}_T}$ when mapping the tuple type [*name : string city : string*].

In order to formally introduce the disjoint concepts $A_C, A_{b_i}, A_{\mathcal{V}_T}$, we extend the concept language syntax

$$C, D \to A \,|\, A_C \,|\, A_{b_i} \,|\, A_{\mathcal{V}_t} \,|\, \top \,|\, C \sqcap D \,|\, \forall_{[m,n]} R.C \,|\, f:C$$

adding the following equations to the semantics:

$$(A_C)^{\mathcal{I}_C} = \Delta^{\mathcal{I}_C} \setminus \left((A_{\mathcal{V}_T})^{\mathcal{I}_C} \cup_i (A_{b_i})^{\mathcal{I}_C}\right)$$
$$(A_{\mathcal{V}_T})^{\mathcal{I}_C} = \Delta^{\mathcal{I}_C} \setminus \left((A_C)^{\mathcal{I}_C} \cup_i (A_{b_i})^{\mathcal{I}_C}\right)$$
$$(A_{b_i})^{\mathcal{I}_C} = \Delta^{\mathcal{I}_C} \setminus \left((A_{\mathcal{V}_T})^{\mathcal{I}_C} \cup (A_C)^{\mathcal{I}_C} \cup_{j \neq i} (A_{b_j})^{\mathcal{I}_C}\right)$$

4.1 Consistency

In order to show that our mapping is consistency-preserving, it is useful to define an *expanded form* for database class declarations. Arbitrary class descriptions can be rewritten as equivalent expanded class descriptions by applying the following *expansion procedure*—EXP(C)—that maintains the equivalence in meaning to the original class declaration.

1) *Elimination of primitive definitions*: Any partial class declaration (i.e., any primitive class definition) is replaced by a complete definition by means of a newly introduced class name. For example, the primitive class:

$$\textbf{class } Person \lesssim [name:string \;\; birthdate:Date]$$

is replaced by the following defined class

$$\textbf{class } Person \doteq \textbf{isa } Person^*[name:string \;\; birthdate:Date]$$

where *Person** is an undescribed class that stands for the missing part of the definition of *Person*.

2) *Expansion of classes in the* **isa** *clause*: Any class present in the **isa** clause is substituted by its description. In particular, we replace each class name with its superclasses and we inherit the labels. This process is iterated until only undescribed class names are present in the **isa** clause. For example, given the class description:

$$\textbf{class } Employee \doteq \textbf{isa } Person[emp-code:string \;\; salary:\mathrm{int}],$$

the *Person* class is replaced by its description so that the *Employee* class is described by

$$\textbf{class } Employee \doteq \textbf{isa } Person^*[emp-code:string \;\; salary:int \;\; name:string \;\; birthdate:Date]$$

3) *Elimination of type names*: Each type name occurring in the definion of classes is substituted by its definition. For example, assuming that:

$$\textbf{type } Date = [day:int \;\; month:string \;\; year:int],$$

we obtain

$$\textbf{class } Employee \doteq \textbf{isa } Person^*[emp-code:string \;\; salary:int \;\; name:string \;\; birthdate:[day:int \;\; month:string \;\; year:int]]$$

This process is iterated until the description of a class only contains class or basic type names.

PROPOSITION 1. *Let C be a database description class, then* $C^{\mathcal{I}_\mathcal{D}} \equiv (EXP(C))^{\mathcal{I}_\mathcal{D}}$ *for each interpretation* $\mathcal{I}_\mathcal{D}$.

At the end of the expansion process, for each class we have the following general *expanded form*:

$$\textbf{class } C \doteq \textbf{isa } C_1^* \ldots C_n^*[l_1:t_{11} \ldots l_1:t_{1k} \;\; l_2:t_{21} \ldots l_2:t_{2k} \ldots l_m:t_{m1} \ldots l_m:t_{mk}] \quad (1)$$

where $C_1^* \ldots C_n^*$ are undescribed class names, and t_{ij} are class names or basic, tuple or set types.

Note that the expansion of our database descriptions differs a great deal from the expansion procedures used in concept languages [18]: Given that classes and tuple types have disjoint denotations we cannot substitute a class name, restricting a label in a tuple, with the corresponding tuple type that describes it.

At this point, we are able to prove the correctness of the mapping.

THEOREM 1. *Let C be a database class description, then C is* satisfiable *iff* $\mathcal{N}[C]$ *is* satisfiable.

PROOF. :'⇐' First we show that if C is unsatisfiable, then $\mathcal{N}[C]$ is also unsatisfiable. Without any loss of generality, let C be in its expanded form, as showed in (1), and let $t_c = [l_1 : t_{11} \ldots l_1 : t_{1k}\; l_2 : t_{21} \ldots l_2 : t_{2k} \ldots l_m : t_{m1} \ldots l_m : t_{mk}]$. The only source of unsatisfiability in the C class description is t_c, the tuple type that describe it. In fact, if t_c is satisfiable then there is an interpretation $\mathcal{I}_\mathcal{D}$ such that $(t_c)^{\mathcal{I}_\mathcal{D}} \neq \emptyset$. Then we can build an interpretation $\overline{\mathcal{I}_\mathcal{D}} = (\overline{\mathcal{V}}, \overline{\delta}, \overline{\cdot^{\mathcal{I}_\mathcal{D}}})$ such that: $(t_c)^{\overline{\mathcal{I}_\mathcal{D}}} = (t_c)^{\mathcal{I}_\mathcal{D}}$, with an object $o \in \overline{\mathcal{O}}$ such that $\overline{\delta}(o) \in (t_c)^{\overline{\mathcal{I}_\mathcal{D}}}$ and $o \in \bigcap_{i=1}^n (C_i^*)^{\overline{\mathcal{I}_\mathcal{D}}}$—because $C_1^* \ldots C_n^*$ are undescribed classes,

3. In the following, we will use *String* and *Int* as undescribed concepts mapping the set of basic values *string* and *integer*.

denoting arbitrary sets of objects. Then $o \in C^{\overline{\mathcal{I}_\mathcal{D}}}$, and C is satisfiable. A tuple type is unsatisfiable iff:

1) For a label l_i there is an unsatisfiable type t_{ij};
2) For a label l_i we have: $l_i : t_{ip}, l_i : t_{iq}, (t_{ip})^{\mathcal{I}_\mathcal{D}} \cap (t_{iq})^{\mathcal{I}_\mathcal{D}} = \varnothing$.

Case 1. We only show the case with t_{ij} unsatisfiable class (the proof of the other cases is similar). Since in this case $\mathcal{N}[l_i : t_{ij}] = f_{l_i} : \mathcal{N}[t_{ij}]$, assuming by induction that $\mathcal{N}[t_{ij}] = A_{ij}$ is unsatisfiable, then $A_{ij}^{\mathcal{I}_\mathcal{C}} = \varnothing$, and

$$(f_{l_i} : A_{ij})^{\mathcal{I}_\mathcal{C}} = \{a \in dom f_{l_i}^{\mathcal{I}_\mathcal{C}} \mid f_{l_i}^{\mathcal{I}_\mathcal{C}}(a) \in A_{ij}^{\mathcal{I}_\mathcal{C}}\} = \varnothing.$$

Then $\mathcal{N}[C]$ is unsatisfiable.

Case 2. We show how, each time that

$$(t_{ip})^{\mathcal{I}_\mathcal{D}} \cap (t_{iq})^{\mathcal{I}_\mathcal{D}} = \varnothing,\ \mathcal{N}[l_i : t_{ip}] \sqcap \mathcal{N}[l_i : t_{iq}]$$

is unsatisfiable.

a) t_{ip} and t_{iq} have different types.

If only one of them is a set type then the same label must be both a relation and a feature; we then obtain unsatisfiability. If both t_{ip} and t_{iq} are not set types, then: $\mathcal{N}[l_i : t_{ip}] \sqcap \mathcal{N}[l_i : t_{iq}] = f_{l_i} : (\mathcal{N}[t_{ip}] \sqcap \mathcal{N}[t_{iq}])$. Assuming that t_{ip} is a class and t_{iq} is a tuple type, then $(\mathcal{N}[t_{ip}])^{\mathcal{I}_\mathcal{C}} \sqsubseteq (A_C)^{\mathcal{I}_\mathcal{C}}$ while $(\mathcal{N}[t_{iq}])^{\mathcal{I}_\mathcal{C}} \sqsubseteq (A_{V_T})^{\mathcal{I}_\mathcal{C}}$, and $(A_C)^{\mathcal{I}_\mathcal{C}} \cap (A_{V_T})^{\mathcal{I}_\mathcal{C}} = \varnothing$; therefore, the thesis is true. The case that t_{ip} or t_{iq} is a basic type name is trivial.

b) Both t_{ip} and t_{iq} are set types: $t_{ip} = \{t'_{ip}\}_{min_{ip}, max_{ip}}$ and $t_{iq} = \{t'_{iq}\}_{min_{iq} max_{iq}}$.

$(t_{ip})^{\mathcal{I}_\mathcal{D}} \cap (t_{iq})^{\mathcal{I}_\mathcal{D}} = \varnothing$, iff: 1) $min_{ip} > max_{iq}$ or $max_{ip} < min_{iq}$; 2) $(t'_{ip})^{\mathcal{I}_\mathcal{D}} \cap (t'_{iq})^{\mathcal{I}_\mathcal{D}} = \varnothing$. In both cases, it's easy to prove the thesis.

c) Both t_{ip} and t_{iq} are tuple types.

$(t_{ip})^{\mathcal{I}_\mathcal{D}} \cap (t_{iq})^{\mathcal{I}_\mathcal{D}} = \varnothing$, iff: 1) either t_{ip} or t_{iq} is unsatisfiable. This case is trivial; 2) t_{ip} and t_{iq} have a common label with an incompatible type. Let $\bar{l}$ be such a label, then:

$$\mathcal{N}[l_i : t_{ip}] \sqcap \mathcal{N}[l_i : t_{iq}] = f_{l_i} : (\mathcal{N}[t_{ip}] \sqcap \mathcal{N}[t_{iq}]) =$$
$$f_{l_i} : (\ldots \sqcap \mathcal{N}[\bar{l} : t_p] \sqcap \mathcal{N}[\bar{l} : t_q] \sqcap \ldots),$$

with $(t_p)^{\mathcal{I}_\mathcal{D}} \cap (t_q)^{\mathcal{I}_\mathcal{D}} = \varnothing$; that corresponds to the hypothesis of case 2. Then, assuming by induction that $(\mathcal{N}[\bar{l} : t_p] \sqcap \mathcal{N}[\bar{l} : t_q])^{\mathcal{I}_\mathcal{C}} = \varnothing$, we have the same situation as in case 1; therefore, the thesis is true.

d) Both t_{ip} and t_{iq} are class names. The proof is similar to the above case.

Thus we have shown that if $\mathcal{N}[C]$ is satisfiable, C is also satisfiable. Now we show that if $\mathcal{N}[C]$ is unsatisfiable, C is unsatisfiable, too.

'$\Rightarrow$' Starting from an expanded class C, the general form of $\mathcal{N}[C] = A$ is:

$$A \doteq A_C \sqcap C_1^* \sqcap \ldots \sqcap C_n^* \sqcap f_1 : A_{f_1} \sqcap \ldots \sqcap f_m : A_{f_m}$$
$$\sqcap \forall_{[m_1, n_1]} R_1 . A_{R_1} \ldots \sqcap \forall_{[m_k, n_k]} R_u . A_{R_k}$$

then A is unsatisfiable iff: i) For a feature f_i (or a role R_i), $A_{f_i}(A_{R_i})$ is unsatisfiable; ii) For $f_i = f_j = \bar{f}$ (or for $R_i = R_j = \bar{R}$), $A_{f_i} \sqcap A_{f_j}$ ($A_{R_i} \sqcap A_{R_j}$) is unsatisfiable; iii) For $R_i = R_j$, then $m_i > n_j$ or $n_i < m_j$. We only sketch the proof for case ii.

Case ii. Let, for $f_i = f_j = \bar{f}$, be $A_{f_i} \sqcap A_{f_j}$ unsatisfiable; then there must be a label $\bar{l}$ in C such that: $[\ldots \bar{l} : t_i\, \bar{l} : t_j \ldots]$, with $\mathcal{N}[t_i] = A_{f_i}$ and $\mathcal{N}[t_j] = A_{f_j}$ and t_i, t_j not set types. If $A_{f_i} \sqcap A_{f_j}$ is unsatisfiable, then for t_i, t_j we have:

a) t_i, t_j have different types. The thesis is trivial.

b) t_i, t_j are class names.

Now, $([\bar{l} : t_i\ \bar{l} : t_j])^{\mathcal{I}_\mathcal{D}} \equiv ([\bar{l} : t'])^{\mathcal{I}_\mathcal{D}}$, with $t' \doteq$ **isa** t_i t_j, but $\mathcal{N}[t'] = A_{f_i} \sqcap A_{f_j}$, and $(\mathcal{N}[t'])^{\mathcal{I}_\mathcal{C}} = \varnothing$, then we have the same situation as in case i. Therefore, the thesis is true.

c) t_i, t_j are tuple types. The proof is similar to the previous case. □

The following corollaries naturally derive from the preceding theorem:

COROLLARY 1. *$\mathcal{N}$ is an isomorphism between the set C of database classes and the set $\mathcal{N}[C]$, with the subsumption as an order relationship. Then $\forall\ C_1, C_2 \in C$:*

1) $C_1 \neq C_2 \Rightarrow \mathcal{N}[C_1] \neq \mathcal{N}[C_2]$;
2) $C_1 \sqsubseteq C_2 \Leftrightarrow \mathcal{N}[C_1] \sqsubseteq \mathcal{N}[C_2]$.

COROLLARY 2. *$\forall\ C_1, C_2 \in C$, then $C_1 \approx C_2$ ($A \approx B$ iff $A \sqsubseteq B \wedge B \sqsubseteq A$)*

$\Leftrightarrow \mathcal{N}[C_1] \approx \mathcal{N}[C_2]$.

5 ASSERTIONAL MAPPING

In the previous section, we discussed the possibility of mapping syntactic descriptions from an object data model to a concept language, i.e., schema descriptions into Tbox descriptions. Here we show how to map a world description, while preserving its consistency, by means of an extension of the syntactic mapping $\mathcal{N}$ over the extensional level of the knowledge base. Before defining the assertional mapping, we briefly sketch the assertional formalism in the two languages.

Let a, b be individual names and C (R, f) be a concept (role, feature); the assertional formalism generally used in concept languages allows us to state that individuals are instances of concepts, and that pairs of individuals are instances of roles or features, by means of the following *assertional axioms*: $a : C$, aRb, afb. An interpretation $\mathcal{I}_C$ *satisfies* the assertional axioms $a : C$ iff $a \in C^{\mathcal{I}_\mathcal{C}}$, aRb iff $(a, b) \in R^{\mathcal{I}_\mathcal{C}}$, afb iff $f^{\mathcal{I}_\mathcal{C}}(a) = b$.

A finite set of assertional axioms is called Abox $\mathcal{A}$. We say that an interpretation $\mathcal{I}_C$ is a *model* of an Abox $\mathcal{A}$ wrt a Tbox $\mathcal{T}$ if $\mathcal{I}_C$ satisfies all the assertional axioms in $\mathcal{A}$ and all the terminological axioms in $\mathcal{T}$; furthermore, an Abox $\mathcal{A}$ is *consistent* wrt a Tbox $\mathcal{T}$ if $\mathcal{A}$ has a model.

The assertional formalism used in the object data model specifies the class that an individual is instance of and the structured value associated with it by means of the assertions $o : C$ and $o : [l_1 : v_1 \ldots l_n : v_n]$. We say that an interpretation $\mathcal{I}_D$ *satisfies* the assertions $o : C$ iff $o \in C^{\mathcal{I}_\mathcal{D}}$ and $o : [l_1 : v_1 \ldots l_n : v_n]$ iff $\delta(o) = [l_1 : v_1 \ldots l_n : v_n]$.

Let a database DB be a finite set of assertions. $\mathcal{I}_D$ is a *model* of DB wrt a schema S iff $\mathcal{I}_D$ satisfies all the descriptions in S and all the assertions in DB; a database DB is *consistent* wrt a schema S if it has a model.

Before defining the assertional mapping, we give the definition of *value mapping* that allows us to build a domain Δ^{I_C} from a generic set of values $\mathcal{V}$.

DEFINITION 3 (Value Mapping): *Let us extend the syntactic mapping $\mathcal{N}$ over the set of values $O \cup \mathcal{B} \cup \mathcal{V}_T$ to the domain Δ^{I_C} so that it is injective and:*

1) $\forall o \in O. \mathcal{N}[o] = o_c,\ o_c \in (A_C)^{I_C}$;
2) $\forall v_{b_i} \in \mathcal{B}_i. \mathcal{N}[v_{b_i}] = o_{b_i},\ o_{b_i} \in (A_{b_i})^{I_C}$;
3) $\forall v_t \in \mathcal{V}_T. \mathcal{N}[v_t] = o_{v_t},\ o_{v_t} \in (A_{V_T})^{I_C}$. *Further, let* $v_t = [l_1 : v_1 \ldots l_n : v_n]$ *then* o_{v_t} *is such that:*
   a) $\forall v_i \notin \mathcal{V}_S. (f_{l_i})^{I_C}(o_{v_t}) = \mathcal{N}[v_i]$
   b) $\forall v_i \in \mathcal{V}_S,\ v_i = \{v_{i1} \ldots v_{im}\}. (o_{v_t}, \mathcal{N}[v_{ik}]) \in (R_{l_i})^{I_C}$, *for* $k = 1, \ldots, m$.

DEFINITION 4 (Assertional Mapping): *In order to define the assertional mapping, we extend the mapping $\mathcal{N}$ so that it associates a corresponding assertional axiom to each database assertion, in such a way that:*

1) $\mathcal{N}[o : C] = \mathcal{N}[o] : \mathcal{N}[C]$;
2) $\mathcal{N}[o : [l_1 : v_1 \ldots l_n : v_n]]$ *is such that:*
   a) $\mathcal{N}[o] : A_C$;
   b) $\forall v_i \notin \mathcal{V}_S. \mathcal{N}[o] f_{l_i} \mathcal{N}[v_i]$;
   c) $\forall v_i \in \mathcal{V}_S,\ v_i = \{v_{i1} \ldots v_{im}\}. \mathcal{N}[o] R_{l_i} \mathcal{N}[v_{ik}]$, *for* $k = 1, \ldots, m$.

Let us consider the following database assertions—for convention, we show `Individual-Names` in typewriter font:

`Alex` : *Student*, `MIT` : *College*

`Alex`: [*name*: *"Alessandro"* *birthdate*: [*day*: 20 *month*: *"July"* *year*: 1954] *regist-num*: 128 *enrolled*: `MIT` *enrolled-course*: {*"Database"*}]

The corresponding Abox assertions are the following:

`Alex`: A_C, `Alex`: *Student*, `MIT`: A_C, `MIT`: *College*, $\texttt{O}_1$: A_{V_T},

`Alessandro`: *String*, `Database`: *String*, `July`: *String*, `20`: *Int*, `1954`: *Int*, `128`: *Int*,

`Alex` *name* `Alessandro`, `Alex` *birthdate* $\texttt{O}_1$, `Alex` *regist-num* `128`,

`Alex` *enrolled* `MIT`, `Alex` *enrolled-course* `Database`,

$\texttt{O}_1$ *day* `20`, $\texttt{O}_1$ *month* `July`, $\texttt{O}_1$ *year* `1954`.

Note that the mapping has introduced new individuals corresponding to each basic value present in the database assertions (e.g., `Alessandro`, `20`, etc.) and the individual $\texttt{O}_1$ belonging to the class A_{V_T} in order to consider the tuple value associated with the *birthdate* label in the database assertions concerning the individual `Alex`.

The following theorem follows from the above definitions and Theorem 1.

THEOREM 2. *Given a schema S, let $\mathcal{T}$ be the Tbox obtained by the $\mathcal{N}$ mapping; analogously, let DB be a set of database assertions and $\mathcal{A}$ the Abox obtained by the $\mathcal{N}$ mapping. DB is consistent wrt S if and only if $\mathcal{A}$ is consistent wrt $\mathcal{T}$.*

6 Concluding Remarks

In this paper, we discuss some similarities and differences existing between object database models and concept languages. In particular, we focus on their characteristics involved in defining objects and values and their mutual relationships. We illustrate how object database descriptions can be transformed into concept language descriptions by a suitable mapping, capable of preserving satisfiability. In our opinion, this correspondence presents a formal framework that can be used for treating common aspects of database systems and knowledge representation systems. In particular, it is possible for object databases to exploit both the algorithms developed in concept language environments for performing subsumption, consistency check, realization and retrieval and the results concerning their complexity. The formal model-theoretic semantics of concept languages provides means for investigating soundness and completeness of inference algorithms. Furthermore, many studies in the concept language community concern the computational complexity of the reasoning tasks offered. With respect to the database model presented, it can be deduced that the subsumption is polynomial (as we have already proven in [2]); at an extensional level, assertion satisfiability and instance checking are also polynomial in the size of the knowledge base [16].

Acknowledgments

This work was partially supported by the Italian National Research Council (CNR), project "Sistemi Informatici e Calcolo Parallelo." We would like to thank Enrico Franconi for the helpful and incisive discussions we had with him.

References

[1] S. Abiteboul and P.C. Kanellakis, "Object Identity as a Query Language Primitive," *ACM SIGMOD*, pp. 170-173,1989.
[2] A. Artale, F. Cesarini, and G. Soda, "Subsumption Computation in an Object-Oriented Data Model," *Proc. Workshop Processing Declarative Knowledge*, vol. 567, *Lecture Notes in Artificial Intelligence*, pp. 124-140. Springer-Verlag,1991.
[3] A. Artale, F. Cesarini, and G. Soda, "Introducing Taxonomic Reasoning in LOGIDATA+," *LOGIDATA+: Deductive Databases with Complex Objects*, P.Atzeni, ed., vol. 701, *Lecture Notes in Computer Science*, pp. 85-104. Springer-Verlag,1993.
[4] F. Baader and B. Hollunder, "A Terminological Knowledge Representation System with Complete Inference Algorithm," *Proc. Workshop Processing Declarative Knowledge*, vol. 567, *Lecture Notes in Artificial Intelligence*. Springer-Verlag,1991.
[5] H.W. Beck, S.K. Gala, and S.B. Navathe, "Classification as a Query Processing Technique in the CANDIDE Data Model," *Proc. of Fifth Int'l Conf. Data Eng.*, pp. 572-581, Los Angeles, 1989.
[6] D. Beneventano, S. Bergamaschi, and B. Nebel, "Subsumption for Complex Objects Data Models," *Proc. Int'l Conf. Database Theory*, Berlin,1992.
[7] S. Bergamaschi and C. Sartori, "On Taxonomic Reasoning in Conceptual Design," *ACM Trans. Database Systems*, vol. 17, Sept. 1992.
[8] R.J. Brachman, V.P. Gilbert, and H.J. Levesque, "An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts in KRIPTON," *Proc. of the Ninth Int'l Joint Conf. Artificial Intelligence*, pp. 532-539, Los Angeles, 1985.
[9] R.J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Resnick, "Living with CLASSIC: When and How to Use a KL-ONE-Like Language," *Principles of Semantic Networks*, J. Sowa, ed. Morgan Kaufmann, 1991.
[10] L. Cardelli, "A Semantics of Multiple Inheritance," *Semantics of Data Types*, pp.51-67. Springer-Verlag, 1984.
[11] R. Cattoni and E. Franconi, "Walking Through the Semantics of Frame-Based Description Languages: A Case Study," *Proc. Fifth Int'l Symp. Metodologies for Intelligent Systems*, Knoxville, Tenn., North-Holland, 1990.
[12] F.M. Donini, B. Hollunder, M. Lenzerini, A. Marchetti Spaccamela, D. Nardi, and W. Nutt, "The Complexity of Existential Quantification in Concept Languages," *Artificial Intelligence*, vol. 53, pp. 309-327,1992.
[13] F.M. Donini, M. Lenzerini, D. Nardi, and W. Nutt, "Tractable Concept Languages," *Proc. 12th Int'l Joint Conf. Artificial Intelligence*, pp. 458-465, Sidney, 1991

[14] B. Hollunder, W. Nutt, and M. Schmidt-Schauß, "Subsumption Algorithms for Concept Description Languages," *Proc. Ninth European Conf. Artificial Intelligence*, pp. 348-353, Stockholm, Aug. 1990.
[15] C. Lecluse and P. Richard, "Modelling Complex Structures in Object-Oriented Databases," *ACM PODS*, pp. 360-367,1989.
[16] M. Lenzerini and A. Schaerf, "Concept Languages as Query Languages," *Proc. Ninth Nat'l Conf. Artificial Intelligence, AAAI-91*, pp. 471-476, Anaheim, Calif.,1991.
[17] R. MacGregor, "Inside the LOOM Description Classifier," *SIGART Bulletin*, vol. 2, pp. 88-92,1991.
[18] B. Nebel, *Reasoning and Revision in Hybrid Representation Systems*, Vol. 422, *Lecture Notes in Artificial Intelligence*. Berlin, Heidelberg, New York: Springer-Verlag, 1990.
[19] C. Peltason, "The BACK Systems—An Overview," *SIGART Bulletin*, vol. 2, pp. 114-119,1991.
[20] M. Schmidt-Schauß and G. Smolka, "Attributive Concept Descriptions with Complements," *Artificial Intelligence*, vol. 48, pp. 1-26,1991.
[21] J.G. Schmolze and W.S. Mark, "The NIKL Experience," *Computational Intelligence*, vol. 6, pp. 48-69,1991.

Correction to a Footnote in "Theoretical and Practical Considerations of Uncertainty and Complexity in Automated Knowledge Acquisition"

Xiao-Jia M. Zhou and Tharam S. Dillon

1 Introduction

IN a footnote on p. 703 of our recent paper [2], we referred to the distance measure by Lopez de Mantaras [1]. The footnote should be corrected as follows:

> Recently, Lopez de Mantaras [1] proposed a distance-based attribute-selection measure as the "proper" normalization for Quinlan's information-gain criterion. It is proved by the contingency subdividing test that this measure is not biased towards attributes with more values.

References

[1] R. Lopez de Mantaras, "A Distance-Based Attribute Selection Measure for Decision Tree Induction," *Machine Learning*, vol. 6, pp. 81-92, 1991.
[2] X.J. Zhou and T.S. Dillon, "Theoretical and Practical Considerations of Uncertainty and Complexity in Automated Knowledge Acquisition," *IEEE Trans. Knowledge and Data Engineering*, vol. 7, no. 5, pp. 699-712, 1995.