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 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 (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 [11], [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 reasoning 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 object 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.
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-id> <class-declaration> | <class-id> <prim-def> isa <class-id> <tuple-type>
<type> ::= <tuple-type> | <class-id> | <type-id> | <basic-type>
<tuple-type> ::= [ <component> ]
<component> ::= <label> <set-type> | <label> : <type>
<set-type> ::= <type> | \{ | \}

<basic-type> ::= string | integer | bool
<prim-def> ::= \{ | \}

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”:

\[
\text{class } \text{Father} \equiv \text{isa } \text{Person} \land \text{child} \quad \text{[Person]}
\]

Given a set of declarations, i.e., a database schema \( S \), an interpretation \( I_S = (\mathcal{V}, \delta, ^I_S) \) consists of:

- A set \( \mathcal{V} \) of values (the domain of \( ^I_S \)), \( \mathcal{V} = \mathcal{B} \cup \mathcal{O} \cup \mathcal{V}_t \cup \mathcal{V}_g \) with:
  1. \( \mathcal{B} = \{^I_{b_i}, b_i\} \), \( b_i \) set of values associated with each basic type; \( b_i \cap b_j = \emptyset, \forall i, j, i \neq j \).
  2. \( \mathcal{O} \) is a countable set of symbols called object identifiers disjoint from \( \mathcal{B} \).
  3. \( \mathcal{V}_t \) is the set of tuple values: \( \mathcal{V}_t = \{v_i | v_i \text{ is a mapping from the set of labels to } \mathcal{V}\} \). We denote by \( t_i : v_i \) the mapping defined on \( [l_1 ... l_k] \) such that \( v_i(l_i) = v_i, v_i(l_i) = 1, ... k \).
  4. \( \mathcal{V}_g \) is the set of set-values: \( \mathcal{V}_g = \{v_i | v_i \subseteq \mathcal{V}\} \). A set-value is denoted by \( [v_i | v_i \subseteq \mathcal{V}] \) such that \( v_i = \mathcal{V}, v_i = 1, ... k \).
- A mapping \( \delta \) that associates a tuple value with each object-identifier: \( \delta : \mathcal{O} \rightarrow \mathcal{V}_t \).
- An interpretation function \( ^{I_S} \) that maps every syntactic constructor to a subset of \( \mathcal{V} \) in such a way that:

\[
\begin{align*}
&\text{(<basic-type>} >^{I_S} \mathcal{B} \\
&\text{(<tuple-type} >^{I_S} = (l_i : v_i, ... l_k : v_k) \mathcal{V}_g \\
&\text{(<set-type} >^{I_S} = (l_i : v_i) \mathcal{V}_g = \{v_i \in \mathcal{V} | m \leq v_i \subseteq \mathcal{V} \leq n \} \\
&\text{(<type-id} >^{I_S} = \text{(<tuple-type} >^{I_S} = \mathcal{V}_g \\
&\text{(<class-id} >^{I_S} = \text{(<tuple-type} >^{I_S} = \mathcal{V}_g \\
&\text{Universal Class} \\
&\text{Defined Class} \\
&\text{Primitive Class}
\end{align*}
\]

For the interpretation of \( \text{Type}(C) \) we have the following recursive definition.

**DEFINITION 1 (Interpretation of \( \text{Type}(C) \): Given the class declarations:**

\[
\text{class } C_0 \equiv \text{prim-def} \equiv \text{tuple-type}_0
\]

\[
\text{class } C_1 \equiv \text{prim-def} \equiv \text{isa } C_0 \equiv \text{tuple-type}_1
\]

\[
\text{... ... ... } \equiv \text{isa } C_n \equiv \text{tuple-type}_n
\]

\[
\text{... ... ... } \equiv \text{isa } C_{n+1} \equiv \text{tuple-type}_{n+1}
\]

\[
\text{we have:}
\begin{align*}
&\text{(<tuple-type}_0 \equiv \text{(<tuple-type} >^{I_S} \text{(<tuple-type}_n \equiv \text{(<tuple-type} >^{I_S} \text{(<tuple-type}_{n+1} \equiv \text{(<tuple-type} >^{I_S} \\
&\text{(<tuple-type}_0 \equiv \text{(<tuple-type} >^{I_S} \text{(<tuple-type}_n \equiv \text{(<tuple-type} >^{I_S} \text{(<tuple-type}_{n+1} \equiv \text{(<tuple-type} >^{I_S}
\end{align*}
\]

This semantics allows us to consider the \( \text{isa} \) 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 \( ^{I_S}(t) \subseteq ^{I_S}(t') \). 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

\[
\text{class } Person \equiv [\text{name: string, birthdate: Date}] \\
\text{class } Project \equiv [\text{proj-code: string, description: string}] \\
\text{class } Student \equiv \text{isa } Person \equiv [\text{registr-num: int, enrolled: } \\
\text{College enrolled: course: string}]
\]

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 registr-num, are enrolled to a College, and are enrolled to some courses. Because of the recursive definition of the Type'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 la-
bol, 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, birth-
date, regist-num, enrolled, and enrolled-course, we can deduce that
this object belongs to Student, too. In general, fixed the interpreta-
tion of the primitive classes, the interpretation of the defined
classes is unambiguously determined.

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

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 sub-
sumption is not explicitly defined by means of an isa clause. Be-
cause 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 da-
tabase 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:

$$C, D \rightarrow A \mid C \rightarrow D \mid \forall_{\text{excl}} R.C \mid f:C$$

An interpretation $I_C = (\Delta_C, \lambda_C)$ consists of a set $\Delta_C$ (the domain
of $I_C$) and a function $\lambda_C$ (the interpretation function of $I_C$) that maps
every concept term to a subset of $\Delta_C$, every role to a subset of $\Delta_C \times \Delta_C$ and every feature to a partial function $f^{I_C}$ from $\Delta_C$ to
$\Delta_C$ (we denote the domain of $f^{I_C}$ as $\text{dom}f^{I_C}$) in such a way that the
following equations are satisfied:

$$\forall C \in D^{I_C} = C^{I_C} \subseteq D^{I_C}$$

$$f^{I_C} = [a \in \text{dom}f^{I_C} \mid f^{I_C}(a) \in C^{I_C}]$$

$$\forall_{\text{excl}} R.C^{I_C} = \{a \in \Delta_C \mid \exists b \in \Delta_C \mid (a, b) \in R^{I_C}\}$$

Features were recently introduced for distinguishing 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 $I_C$ is a model for a concept term C if $C^{I_C} \neq \emptyset$. 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 $C \subseteq D$) if $C^{I_C} \subseteq D^{I_C}$ for every interpretation $I_C$. Sub-
summation can be reduced to satisfiability since C is subsumed by D
if and only if $C \cap D$ is not satisfied.

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 $A \subseteq C$ and $A \equiv C$. An interpretation $I_C$ satisfies $A \subseteq C$
if $A^{I_C} \subseteq C^{I_C}$, while it satisfies $A \equiv C$ if $A^{I_C} = C^{I_C}$. Furthermore,
we denote as an undescribed concept an atomic concept that never
appears as the first argument of a terminological axiom. A termi-
нологy $T$ is a finite set of terminological axioms with the additional
restriction that

1) every atomic concept may appear only once as the first ar-
gument of a terminological axiom in T, and
2) T must not contain cyclic definitions.

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

$$\forall x (x \in \text{Sex} \rightarrow \exists y (y \in \text{Sex} \land \text{Father}(x, y)))$$

$$\forall x (x \in \text{Sex} \rightarrow \exists y (y \in \text{Sex} \land \text{Mother}(x, y)))$$

express that:

1. A Male is a Person;
2. A Father 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 Mother is exactly a Person with at least one and at most
three children 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 con-
junction of concepts, while each tuple type gives rise to a conjunc-
tion 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 disjunc-
tiveness between classes, basic types and tuple types.

DEFINITION 2 (Syntactic Mapping): Let $N$ be a function from database
class declarations to terminological axioms. Given the database
class declaration: class C <prim-def> isa C_1, ..., C_n; l_1, ..., l_m; then
$N<$class-declaration$>$1 is the terminological axiom:

$$A < \text{prim} - \text{def} > A_1 \land A_2 \land \ldots \land A_n$$

where the following equations hold:

$$N[l_i; t_i] = f_i : \text{N}[t_i]; \text{if } t_i \text{ is not a set type}$$

$$N[l_i; t_i] = \forall_{\text{excl}} R_i \cdot N[t_i]; \text{if } t_i = [f_i]_{n,m}$$

For $N[t_i]$ we have:

1. Let $t_i$ be a basic type, then: $N[t_i] = N[t_i] = A_0$
2. Let $t_i$ be a class name, then: $N[t_i] = N[C_i] = A_i$
   and if $A_i$ is a concept then $A_i \subseteq A_i$
3. Let $t_i = [l_1; t_1, l_2; t_2, ..., l_m; t_m]$, then:

2. Although $C \cap D$ 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).
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.

3. In the following, we will use `String` and `Int` as undescended concepts mapping the set of basic values `string` and `integer`.
denoting arbitrary sets of objects. Then \( o \in C^D \), and \( C \) is satisfiable. A tuple type is unsatisfiable iff:

1. For a label \( l \) there is an unsatisfiable type \( t_l \);
2. For a label \( l \), we have \( l; t_{p_l} l; t_{q_l} \subseteq \{ t_{(p_l)} \} \cap \{ t_{(q_l)} \} = \emptyset \).

**Case 1.** We only show the case with \( t_l \) unsatisfiable class (the proof of the other case is similar). Since in this case \( \mathcal{N}[l; t_l] = f_{l}; \mathcal{N}[l,t_l] \), assuming by induction that \( \mathcal{N}[t_l] = A_t \) is unsatisfiable, then \( A_t^k \) is \( \emptyset \), and

\[
\{ t_{(p_l)} \} \cap \{ t_{(q_l)} \} = \{ a \in dom_{t_{(p_l)}}^k \mid f_{(a)}^k (a) \in A_t^k \} = \emptyset.
\]

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

**Case 2.** We show how, each time that

\[
\{ t_{(p_l)} \} \cap \{ t_{(q_l)} \} = \emptyset, \quad \mathcal{N}[l; t_{p_l}] \cap \mathcal{N}[l; t_{q_l}]
\]

is unsatisfiable.

a) \( t_p \) and \( t_q \) 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 \( t_p \) and \( t_q \) are not set types, then: \( \mathcal{N}[l; t_{p_l}] \cap \mathcal{N}[l; t_{q_l}] = f_{l} ; (\mathcal{N}[t_{p_l}] \cap \mathcal{N}[t_{q_l}]). \) Assuming that \( t_p \) is a class and \( t_q \) is a tuple type, then \( (\mathcal{N}[t_{p_l}]^k) \subseteq (A_t^k) \) and \( (\mathcal{N}[t_{q_l}]^k) \subseteq (A_t^k) \), and \( (A_t^k) \cap (A_t^k) = \emptyset \); therefore, the thesis is true. The case that \( t_p \) or \( t_q \) is a basic type name is trivial.

b) Both \( t_p \) and \( t_q \) are set types:

\[
\{ t_{(p_l)} \} \cap \{ t_{(q_l)} \} = \emptyset, \quad \text{iff 1:} \ min_y > \max_y \text{ or } \max_y < \min_y;
\]

\[
\{ t_{(p_l)} \} \cap \{ t_{(q_l)} \} = \emptyset, \quad \text{In both cases, it's easy to prove the thesis.}
\]

c) Both \( t_p \) and \( t_q \) are tuple types.

\[
\{ t_{(p_l)} \} \cap \{ t_{(q_l)} \} = \emptyset, \quad \text{iff 1:} \ \text{either } t_p \text{ or } t_q \text{ is unsatisfiable. This case is trivial; 2: } t_p \text{ and } t_q \text{ have a common label with an incompatible type. Let } \mathcal{I} \text{ be such a label, then:}
\]

\[
\mathcal{N}[l; t_{p_l}] \cap \mathcal{N}[l; t_{q_l}] = f_{l} ; (\mathcal{N}[t_{p_l}] \cap \mathcal{N}[t_{q_l}]) = f_{l} ; (\dots \cap \mathcal{N}[\mathcal{I}; t_{p_l}] \cap \mathcal{N}[\mathcal{I}; t_{q_l}] \cap \dots),
\]

with \( \{ t_{(p_l)} \} \cap \{ t_{(q_l)} \} = \emptyset \); that corresponds to the hypothesis of case 2. Then, assuming by induction that \( (\mathcal{N}[t_{p_l}] \cap \mathcal{N}[t_{q_l}])^k = \emptyset \), we have the same situation as in case 1; therefore, the thesis is true.

d) Both \( t_p \) and \( t_q \) 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.

\[=\]

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

\[
A = A_C \cap C_1 \cap \ldots \cap C_n \cap A_{1} \cap \ldots \cap A_{m} \cap \mathcal{N}[n_{1}, n_{2} \ldots n_{k}].
\]

Then \( A \) is unsatisfiable iff:

i) For a feature \( f \) (or a role \( R \), \( A_{f} \cap A_{R} \) is unsatisfiable; ii) For \( f_{l} = f_{l} = f \) (or for \( R = R \cap R \)), \( A_{f} \cap A_{R} \cap A_{R} \) is unsatisfiable; iii) For \( R = R \cap R \) then \( m > n \) or \( n < m \). We only sketch the proof for case ii.

**Case ii.** Let, for \( f_{l} = f_{l} = f \), be \( A_{f} \cap A_{R} \) unsatisfiable, then there must be a label \( \mathcal{I} \) in \( C \) such that \( \{ \mathcal{I}, l, t_{i}, t_{j}, \ldots \} \), with \( \mathcal{N}[I, l, R] = A_{f} \cap A_{R} \cap A_{R} \), and \( t_{i}, t_{j}, \ldots \) not set types. If \( A_{f} \cap A_{R} \) 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, \( \{ \mathcal{I}, t_{i}, l_{j}, t_{j}, \ldots \} \cup \{ \mathcal{I}, t_{j}, l_{i}, t_{j}, \ldots \} \), with \( \mathcal{N}[I, t_{i}, R] = A_{f} \cap A_{R} \cap A_{R} \), and \( \mathcal{N}[I, t_{j}, R] = \emptyset \), 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} \subseteq C_{2} \Rightarrow \mathcal{N}[C_{1}] \subseteq \mathcal{N}[C_{2}].

**Corollary 2.** \( \forall C_{1}, C_{2} \in C \), then \( C_{1} \sim C_{2} \) (\( A \sim B \iff \mathcal{A} \subseteq B \), \( B \subseteq \mathcal{A} \))

\[=\]

\[\mathcal{N}[C_{1}] \sim \mathcal{N}[C_{2}].\]

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: \( a : C \), \( aRb \), \( a \# b \).

An interpretation \( I \) satisfies the assertional axioms \( a : C \) iff \( a \in C^{I} \), \( aRb \) iff \( (a, b) \in R^{I} \), \( a \# b \) iff \( f_{(a)}(a) \neq b \).

A finite set of assertional axioms is called Abox \( \mathcal{A} \). We say that an interpretation \( I \) of a model of an Abox \( \mathcal{A} \) wrt a Tbox \( \mathcal{T} \) if \( I \) 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 \( o \subseteq [t_{1}, v_{1}, \ldots t_{n}, v_{n}] \). We say that an interpretation \( I \) of \( o : C \) iff \( o \subseteq C^{I} \) of \( o \subseteq C^{I} \) and \( o \subseteq [t_{1}, v_{1}, \ldots t_{n}, v_{n}] \) iff \( (t_{1}, v_{1}, \ldots t_{n}, v_{n}) \).

Let a database DB be a finite set of assertions. \( I \) is a model of DB wrt a schema \( S \) iff \( I \) 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 \( D^C \) from a generic set of values \( V \).

**Definition 3 (Value Mapping):** Let us extend the syntactic mapping \( N \) over the set of values \( U \cup \mathcal{B} \cup \mathcal{V}_p \) to the domain \( D^C \) so that it is injective and:

1. \( \forall v \in O \cdot N[v] = o_v, o_v \in (A_v)^C \);
2. \( \forall \psi_i \in \mathcal{B} \cdot N[\psi_i] = \psi_i, \psi_i \in (A_v)^C \);
3. \( \forall \alpha \in \mathcal{V}_p \cdot N[\alpha] = \alpha, \alpha \in (A_v)^C \).

Further, let \( v_i = (l_1, v_1, \ldots, l_m, v_m) \) then \( v_i \) is such that:

a) \( \forall j_i \not\equiv \mathcal{V}_p, (j_i)^C(\psi_i) = N[v_j] \)

b) \( \forall j_i \not\equiv \mathcal{V}_p, v_i = (v_{i1}, \ldots, v_{im}), N[\psi_i] \in (K_i)^C, \) for \( k = 1, \ldots, m \).

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

1. \( N[O \cdot C] = N[o] \cdot N[C] \);
2. \( N[O_i \cdot l_1, v_1, \ldots, l_m, v_m] \) is such that:
   a) \( N[O_i] = A_v \)
   b) \( \forall \psi_i \not\equiv \mathcal{V}_p, N[\psi_i] \in (A_v)^C \).
   c) \( \forall \psi_i \not\equiv \mathcal{V}_p, v_i = (v_{i1}, \ldots, v_{im}), N[\psi_i] \in (K_i)^C, \) 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

**Alessandro**: birthdate = [day: 20 month: "July" year: 1954], regis-num = 128 enrolled = MIT enrolled-course = ["Database"]

The corresponding Abox assertions are the following:

**Alex**: A0, Alex: Student, MIT = A0, MIT = College, O1 = A0,

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

**Alex name Alessandro, Alex birthdate O1, Alex regis-num 128, Alex enrolled MIT, Alex enrolled-course Database, O1 day 20, O1 month July, O1 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 O1 belonging to the class A0, 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 T be the Tbox obtained by the N-mapping; analogously, let DB be a set of database assertions, and A the Abox obtained by the N-mapping. DB is consistent wrt S if and only if A is consistent wrt 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 soundness. 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 Parallello.” We would like to thank Enrico Franconi for the helpful and incisive discussions we had with him.

**References**


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