

Similarity and Analogy over Application Domains

Simone D.J. Barbosa, Karin K. Breitman, Antonio L. Furtado, Marco A. Casanova

Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de S. Vicente, 225 – Rio de Janeiro, Brasil – CEP 22451-900

{simone, karin, furtado, casanova}@inf.puc-rio.br

***Abstract.** Databases, particularly when storing heterogeneous, sparse semi-structured data, tend to provide incomplete information and information which is difficult to categorize. This paper first considers how to classify entity instances as members of entity classes organized in a lattice-like generalization/specialization hierarchy. Then, it describes how the frame representation employed for instances and classes, as well as the closeness criterion involved in the classification method, favors the practical use of similarity and analogy, where similarity refers to instances within the same class, and analogy involves different classes. Finally, the paper argues that similarity and analogy facilitate querying semi-structured data.*

1. Introduction

Considering a set of entity classes, organized in a specialization/generalization lattice-like structure, we first address in this paper the *frame classification problem*, which consists of determining to which class a given instance belongs. Performing this task may not be trivial, when only a few properties of the instance are known, none of which explicitly indicating how to classify the instance. This is often the case for databases storing sparse, semi-structured data. To address this problem, we introduce a frame representation for instances and classes, and define a classification process based on a *closeness criterion* that takes into account the known property values an instance has, as well as how classes are structured.

As a consequence of the discussion about classification, we observe that the closeness criterion employed to match instance frames against class frames can be modified to measure *similarity* with respect to arbitrary frames, thus extending the power to perform queries over the available information. A second notion, namely *analogy* [Holyoak & Thagard 1996], is also brought in, which allows to handle queries wherein connections across different application domains are used to enable still other extended query features. The distinction we establish between similarity and analogy is merely used for the present explanatory purposes. Similarity will refer to instances within the same class, whereas analogy will involve different classes. Gentner and Markman (1997) present a different, but entirely compatible way, to express the distinction.

We note at this point that we use frames only as a convenient notation for collecting all known properties of an instance and for representing the attributes and binary relationships of a class. Our classification process does not rely on any of the (classical) deductive apparatus for frames [Nardi & Brachman 2003].

The frame classification problem we focus on is akin to the classification problem discussed in the context of Description Logic (DL) [Baader & Nutt 2003]. We therefore briefly sketch how frames can be formalized in DL and show that testing for frame subsumption can be performed in time proportional to $n \log n$. However, the classification process we introduce is based on a closeness criterion that is quite different from concept subsumption, the basis of the classification process offered by DL systems. Our closeness criterion is also quite distinct from that described in [Fankhauser et. al. 1991], which is based on fuzzy sets.

The paper is organized as follows. Section 2 introduces frames. Classification across the class hierarchy is the object of section 3. Extended queries employing similarity and analogy are treated, respectively, in sections 4 and 5. Section 6 contains concluding remarks. The annex summarizes a minimal set of concepts from Description Logic.

2. Frames

2.1 Informal definition of frames in the context of the ER Model

An *application domain* can be characterized, in terms of the Entity-Relationship model [Batini et al. 1992], by specifying the properties of the intended *classes* of entities. In the present discussion, for simplicity, we shall only consider attributes and binary relationships as properties of entities. For each class of entities, one privileged attribute will serve to uniquely identify the *instances* of the class. (Another possibility, not to be elaborated here, is to introduce a special kind of property known as an object identifier).

An instance of a given class can be described by indicating its attribute values, and the values of the identifying attributes of the instances associated with it through binary relationships. To represent all currently known properties of an instance of some class, a frame structure [Schank & Colby 1973] is particularly convenient.

A *slot* is an expression of the form “ $P:V$ ” or of the form “ $P:$ ”, where P and V , called the *slot name* and the *slot value*, satisfy one of the following conditions:

- P is an attribute of the entity being described, and V , if defined, is a single value (the attribute is single-valued, by assumption), or
- P is of the form R/I , where R is a binary relationship in which the entity is the first participant, and V , if defined, is a single value or a set of values (the relationship is non-total and multi-valued, by assumption), or
- P is of the form $R/2$, where R is a binary relationship in which the entity is the second participant, and V , if defined, is a single value or a set of values (the relationship is non-total and multi-valued, by assumption)

A *frame* is a set of *slots* with distinct names. The *top frame* is the empty set. An *instance frame* is a frame whose slots are all of the form “ $P:V$ ”, and a *class frame* is a frame with at least one slot of the form “ $P:$ ”.

Consider a relationship `works` (E, C) that associates employees with companies, in this order. Examples of instance frames follow:

```
[name:John, age:35, works/1:ACME, salary:200]
[acronym:ACME, works/2:{Mary, John, Paul}]
[name:Peter, age:20, area:engineering, level:undergraduate]
```

Examples of class frames are (listed with the related classes for later reference):

```
[name:, age:, works/1:, salary:] (of class employee)
[acronym:, branch:, headquarters:, works/2:] (of class company)
[name:, age:, area:, level:] (of class student)
[name:, age:, area:, level:grad, advisor:] (of class grad_student)
[name:, age:, area:law, level:] (of class law_student)
```

Note that the last two class frames include slots with values.

2.2 Formal definition of frames in the context of DL

Let \mathcal{L} be an attributive language with alphabet \mathcal{A} (see the annex). Recall that a concept name in \mathcal{A} denotes a set of individuals (an entity set, using the ER model jargon), and that a role name in \mathcal{A} denotes a binary relation between individuals (and represents either an attribute or a binary relationship, using the ER model jargon, without indicating however which are the entity sets involved, thereby requiring the inclusion of an additional atomic concept C in the definition that follows).

We first introduce additional concept descriptions that do not extend the power of attributive languages, but just simplify the syntax of slots. Let R be a role name, C be a concept name and V, V_1, \dots, V_k be constants. Then, the following expressions are also concept descriptions:

- i) “ $R:C$ ”, “ $R/1:C$ ” and “ $R/2:C$ ”, called *role restrictions*
- ii) “ $R:C:\{V\}$ ”, “ $R/1:C:\{V_1, \dots, V_k\}$ ” and “ $R/2:C:\{V_1, \dots, V_k\}$ ”, called *role value restrictions*

An interpretation function I , with domain \mathcal{D} , is extended to these concept descriptions as follows: (1a) $(R:C)^I$ is the set of individuals s such that, if R^I associates s with an individual t , then t is in C^I , and there is at most one such individual; (1b) $(R/1:C)^I$ is the set of individuals s such that, if R^I associates s with an individual t , then t is in C^I ; (1c) $(R/2:C)^I$ is the set of individuals s such that, if $(R^-)^I$ associates s with an individual t , then t is in C^I ; (2a) $(R:C:\{V\})^I$ is the set of individuals that R^I associates with exactly the individual V^I , which must be in C ; (2b) $(R/1:C:\{V_1, \dots, V_k\})^I$ is the set of individuals that R^I associates with exactly the individuals V_1^I, \dots, V_k^I , which must be in C ; (3b) $(R/2:C:\{V_1, \dots, V_k\})^I$ is the set of individuals that $(R^-)^I$ associates with exactly the individuals V_1^I, \dots, V_k^I , which must be in C . We can then prove that:

- i) $R:C \equiv (\forall R.C \sqcap \leq 1.R)$
- ii) $R/1:C \equiv \forall R.C$
 $R/2:C \equiv \forall R^- . C$

- iii) $R:C:\{V\} \equiv (\forall R.C \sqcap \leq 1.R \sqcap \exists R.\{V\})$
 $R/1:C:\{V_1,\dots,V_k\} \equiv (\forall R.C \sqcap \exists R.\{V_1\} \sqcap \dots \sqcap \exists R.\{V_k\} \sqcap \forall R.\{V_1,\dots,V_k\})$
 $R/2:C:\{V_1,\dots,V_k\} \equiv (\forall R^-.C \sqcap \exists R^-. \{V_1\} \sqcap \dots \sqcap \exists R^-. \{V_k\} \sqcap \forall R.\{V_1,\dots,V_k\})$

A *slot* of \mathcal{L} is a role restriction or a role value restriction in \mathcal{L} . A *frame* of \mathcal{L} is an expression of the form $[F_1,\dots,F_m]$, where F_i is a slot of \mathcal{L} , for each $i \in [1,m]$. The special symbols \top and \perp are also frames, called the *top* and the *bottom frames*, respectively. An *instance frame* is a frame whose slots are all role value restrictions. A *class frame* is a frame with at least one slot which is not a role value restriction. We treat frames as defined notions, in the sense that a frame $[F_1,\dots,F_m]$ is equivalent to the concept description $F_1 \sqcap \dots \sqcap F_m$. Thus, given an interpretation I for \mathcal{L} , we have $[F_1,\dots,F_m]^I = (F_1 \sqcap \dots \sqcap F_m)^I = F_1^I \cap \dots \cap F_m^I$.

A *frame axiom* of \mathcal{L} is an expression of the form $c[F_1,\dots,F_m]$, where c is a concept name of \mathcal{L} and $[F_1,\dots,F_m]$ is a class frame of \mathcal{L} . The special symbols TRUE and FALSE are also frame axioms, called the *top* and *bottom frame axioms*, respectively. We also treat frame axioms as defined notions, in the sense that $c[F_1,\dots,F_m]$ is logically equivalent to the inclusion axiom $c \sqsubseteq F_1 \sqcap \dots \sqcap F_m$ in \mathcal{L} . Therefore, given an interpretation I for \mathcal{L} , we say that I *satisfies* $c[F_1,\dots,F_m]$ iff $c^I \subseteq F_1^I \cap \dots \cap F_m^I$. We note that what we call a frame axiom has no connections with the frame axioms used in the context of planning [Brachman and Levesque 2004].

3. Classification

3.1 Lattice-like structures

Given a set of entity classes C_1,\dots,C_n , with frames F_{C_1},\dots,F_{C_n} , respectively, and an instance I , with frame F_I , the *frame classification problem* consists of determining to which entity class C_k the instance I belongs. Performing this task may be trivial, if a declaration to the effect that $I \in C_k$ is explicitly registered in the database. It is still not too hard if F_I , as extracted from the database, covers all properties specified in F_{C_k} . However, not all such properties may be known at a given time. Furthermore, entity classes are often organized in a specialization/generalization lattice-like structure, via the is-a relation, thus making classification considerably more complex.

Recall that complete lattices include a top (\top) and a bottom (\perp) node and come equipped with two dual basic operators, Δ (meet) and ∇ (join) [MacLane & Birkhoff 1967]. For every pair of nodes N_1 and N_2 , a node $M = N_1 \Delta N_2$ and a node $J = N_1 \nabla N_2$ must exist and be unique. For lattices of entity classes, in order to fulfill this uniqueness requirement, we could be forced to add a number of classes of no practical significance; to avoid this, we shall not insist on uniqueness, thereby the use of the term ‘lattice-like’.

To organize frames in a lattice-like structure, we proceed as follows. We say that a slot r is *subsumed* by a slot s , denoted $r \sqsubseteq s$, iff $r=s$ or r is of the form “ $P:V$ ” and s of the form “ $P:$ ”. We say that r *conflicts with* s iff r and s have the same name, they both have values, but their values are different. We say that a frame F is *subsumed* by a frame G ,

denoted $F \sqsubseteq G$, iff, for every slot $f \in F$, there is $g \in G$ such that $f \sqsubseteq g$. We say that F *conflicts with* G iff there are $f \in F$ and $g \in G$ such that f and g conflict.

Given two frames, F and G , we define $F \Delta G$ as the frame M such that a slot $s \in M$ iff

- (1) $s \in F$ and there is $g \in G$ such that $g \sqsubseteq s$, or
- (2) $s \in G$ and there is $f \in F$ such that $f \sqsubseteq s$

Note that M can be the empty frame, that is, the top frame, when there is no slot satisfying one of the above conditions. Also note that, by either (1) or (2), if $s \in F \cap G$ then $s \in M$, since $s \sqsubseteq s$ always holds, by definition of subsumption.

Let F and G be two frames that do not conflict. We define $F \nabla G$ as the frame J such that a slot $s \in J$ iff one of the following conditions holds:

- (1) $s \in F$ and there is a slot $g \in G$ such that $s \sqsubseteq g$, or
- (2) $s \in G$ and there is a slot $f \in F$ such that $s \sqsubseteq f$, or
- (3) $s \in F$ and there is no slot $g \in G$ such that $g \sqsubseteq s$, or
- (4) $s \in G$ and there is no slot $f \in F$ such that $f \sqsubseteq s$

If F and G conflict, then $F \nabla G$ is undefined. Again note that, by either (1) or (2), if $s \in F \cap G$ then $s \in J$, since $s \sqsubseteq s$ always holds, by definition of subsumption.

At this point, note that the meet (and the join) operation on frames is associative, which precludes parenthesizing expressions composed of multiple meets (or joins).

Consider the following example involving instance frames:

$$[\text{age:25}] = [\text{age:25, salary:100}] \Delta [\text{age:25, area:chemistry}]$$

$$[\text{age: 25, salary:100, area:chemistry}] =$$

$$[\text{age:25, salary:100}] \nabla [\text{age:25, area:chemistry}]$$

Observe the two frames on the right-hand side of the equalities. Computing the meet essentially means to retain only what is shared, in this case the single slot `age:25`. By contrast, computing the join means passing down all slots, common or not; but the common slots must agree with respect to the assigned values: two different ages, say 25 and 30, would conflict, causing the join to be undefined.

The specialization/generalization class hierarchy cannot be defined independently of class frames, however. For example, if classes `employee` and `student` are generalized to class `person`, then the frame for `person` should be the meet of the frames for classes `employee` and `student`. Conversely, if classes `employee` and `student` are specializations of class `person`, then the frames for `employee` and `student` should inherit all slots of the frame for `person`. If some instances of `person` can be both instances of `employee` and of `student`, an even more specialized class, `trainee`, can be added to the hierarchy, thereby establishing a lattice-like structure.

In general, given a set of classes $C=\{C_1,\dots,C_n\}$, with frames $F=\{F_1,\dots,F_n\}$, and a set of is-a relationships between these classes, if C_{i1},\dots,C_{ik} are all classes in C that generalize to C_{ip} , then the frame F_{ip} of C_{ip} must subsume the meet of F_{i1},\dots,F_{ik} . That is, $(F_{i1}\Delta\dots\Delta F_{ik})\sqsubseteq F_{ip}$. Conversely, if C_{i1},\dots,C_{ik} are all classes in C that specialize to C_{ip} , then F_{ip} must be subsumed by the join of F_{i1},\dots,F_{ik} . That is, $F_{ip}\sqsubseteq (F_{i1}\nabla\dots\nabla F_{ik})$.

For example, consider two classes, `employee` and `student`, which generalize to class `person` and specialize to class `trainee`. Assume that their frames are:

```

person: [name:, age:]
employee: [name:, age:, works/1:, salary:]
student: [name:, age:, area:, level:, fee:]
trainee: [name:, age:, works/1:, salary:, area:, level:, fee:]

```

Then, the is-a hierarchy and the frames are consistently defined. Indeed, observe that:

```

[name:, age:] = [name:, age:, works/1:, salary:] Δ
                [name:, age:, area:, level:, fee:]

[name:, age:, works/1:, salary:, area:, level:, fee:] =
                [name:, age:, works/1:, salary:] ∇
                [name:, age:, area:, level:, fee:]

```

Let us now consider a different situation. Suppose that all that is recorded in the database about a certain instance, `Bertillon`, is collected in the instance frame:

```

FB=[area:engineering, fee:45]

```

Compare F_B with the class frame F_E for the class `engineering`:

```

FE=[area: engineering, fee:, gender:, lodging:,
      monthly_pay:, name:, restaurant:, room_phone:, scholarship:]

```

Note that F_E shows a considerably larger number of slots than F_B . But it is clear that F_B is subsumed by F_E . Intuitively, all slots of F_B are accounted for in F_E , and there are no conflicting values in the two frames. Indeed, slot `fee:45` of F_B is subsumed by slot `fee:` of F_E , whereas slot `area:engineering` of F_B also occurs in F_E .

3.2 Classification with incomplete information

Matching instance frames against class frames can thus be regarded as the central task in the frame classification process. However, as we discuss in the sequel, we still have to cope with the incompleteness of the information available about a given instance, and we must devise some suitable criterion for choosing among candidate classes.

Returning to the example at the end of Section 3.1, we saw that `Bertillon` might well be classified as an `engineering` instance. Informally speaking, all properties of `Bertillon` known to us (until this instant of time) are accounted for, and he has no properties that are lacking in, or conflicting with, the specification of `engineering`.

But, instead of engineering, would it be wrong to classify Bertillon as a trainee? Yes, we could call him a trainee if we knew that he had other properties – such as salary, for example – but right now we do not know that, and therefore calling him a trainee would be, for the time being, an unwarranted conclusion, although a revision may be called for later, if additional data ever becomes available. Could we call him simply a student? Yes, but then we would not take advantage of the information we have about his study area. On the other hand, we could not call him a law student as otherwise there would be a value conflict (by definition, all attributes are single-valued).

Informally speaking, we base our classification method, whereby we try to fit an instance frame F_I of an instance I against a class frame F_C of a class C , on a threefold closeness criterion (a_1, a_2, a_3) , where

Closeness criterion:

- a) a_1 is the number of *slots shared* by F_I and F_C
- b) a_2 is the number of *value hits*, i.e., the cases wherein slots in F_C require restricted values, which happen to be present in F_I
- c) a_3 is the *depth* of C , i.e., the level of C in the lattice of entity classes

Let F_E be the frame for class engineering, and F_B be the instance frame for Bertillon, where

```
 $F_B = [\text{area: engineering, fee: 45}]$ 
```

```
 $F_E = [\text{area: engineering, fee:, name:, gender:, lodging:,}$   
           $\text{monthly_pay:, restaurant:, room_phone:, scholarship:}]$ 
```

Assume that engineering is at level 3 of the class hierarchy (below student, which is in turn below person). Then, the closeness of F_B to F_E will be $[2, 1, -3]$.

As might be expected, larger values for a_1 and a_2 increase the fitness. By contrast, a_3 is to be taken inversely to indicate that going further down in the lattice is unjustified when neither a_1 nor a_2 can be improved. To compare the fitness of a given instance I to two candidate classes, A and B , we compare the closeness of I to A and B lexicographically. That is, A will be a better fit than B whenever $(a_1, a_2, a_3) > (b_1, b_2, b_3)$, where (a_1, a_2, a_3) and (b_1, b_2, b_3) are the closeness of I to A and B .

So, in accordance with the closeness criterion, Bertillon is classified as an engineering instance. Yet, we admit that the slots of F_B are compatible with the two other classes, student and trainee. The same cannot be said of class law_student, due to the value conflict with respect to the area attribute. Note that, because our lattice-like structures do not ensure that the result of classification is unique, since more than one equally eligible class may be found for a given instance.

In more detail, our classification algorithm tries to match an instance frame F_I against a class frame F_C as follows. It visits classes in breadth-first order, in view of criterion (c) for closeness. The process goes down in the lattice until finding classes sharing all slots

indicated in F_I (perfect satisfaction of criterion (a) – which is true if the result of the meet of F_I and F_C retains all slots of F_I). The process also makes sure that there are no value conflicts with the class slots – which is true if the join of F_I and F_C does not fail. Having found such classes, the process looks for (possible) specializations that can offer the maximum number of value hits with F_I (criterion (b)), in an attempt to refine the classification, which may carry the process further down in the lattice. Only one class is finally retained, although, as remarked, there may be more than one optimal solution.

3.3 Formalization of the frame classification problem in DL

The notions of slot and frame subsumption are exactly as for concept descriptions of attributive languages. Given two frames F and G of \mathcal{L} , the *meet* of F and G , denoted $F\Delta G$, and the *join* of F and G , denoted $F\nabla G$, are defined as in section 3.1. We can prove that (proofs are omitted for brevity):

Proposition 1: The set of frames of \mathcal{L} , with subsumption \sqsubseteq as the ordering relation, Δ and ∇ as the meet and join operations, respectively, and the special symbols \top and \perp as the top and bottom frames, respectively, is a lattice.

Now, given two frame axioms $a[E]$ and $b[F]$ of \mathcal{L} , the *meet* of $a[E]$ and $b[F]$, denoted $a[E]\wedge b[F]$, is the frame axiom $c[G]$ such that $c\sqsubseteq a\sqcap b$ and $G=E\Delta F$. The *join* of $a[E]$ and $b[F]$, denoted $a[E]\vee b[F]$, is the frame axiom $c[G]$ such that $c\sqsupseteq a\sqcup b$ and $G=E\nabla F$.

Proposition 2: The set of frame axioms of \mathcal{L} , with logical implication \models as the ordering relation, \wedge and \vee as the meet and join operations, respectively, and the special symbols TRUE and FALSE as the top and bottom frame axioms, respectively, is a lattice.

The lattice of concept expressions and the lattice of frames of \mathcal{L} interact in an interesting way, captured by the following proposition.

Proposition 3: Let $a[E]$ and $b[F]$ be two frame axioms of \mathcal{L} . Then, we have:

- i) $a[E], b[F], a \sqsubseteq c, b \sqsubseteq c$ and $G \sqsubseteq (E \Delta F)$ logically implies $c[G]$
- ii) $a[E], b[F], c \sqsupseteq a, c \sqsupseteq b$ and $(E \nabla F) \sqsubseteq G$ logically implies $c[G]$

Proposition 3 formalizes the familiar idea that “as one moves up in the hierarchy of classes, one should take the intersection of the attribute lists. Conversely, as one moves down in the hierarchy of classes, one should take the union of the attribute lists”. Indeed, given two frame axioms $a[E]$ and $b[F]$, to move up in the class hierarchy means to select an atomic concept c such that $a \sqsubseteq c$ and $b \sqsubseteq c$. The argument for “moving down the class hierarchy” is entirely similar.

The frame classification problem is a variation of the classification problem for DL [Baader and Nutt 2003]: “Let \mathbf{G} be a set of frames of \mathcal{L} and F be a frame of \mathcal{L} . Find the frame $G \in \mathbf{G}$ such that $F \sqsubseteq G$ and there is no frame $H \in \mathbf{G}$ such that $F \sqsubseteq H \sqsubseteq G$ ”. The classification problem therefore requires testing if $F \sqsubseteq G$ holds. Because frames are equivalent to simple, restricted concept descriptions, a variation of the subsumption

structural algorithm [Franconi 2002], developed for the dialect \mathcal{FL}^- of structural DL, can be used to test for frame subsumption in $n \log n$ time.

Algorithm 1: $\text{FRAMESUBS}[F, G]$

Input: $F=[F_1, \dots, F_m]$ and $G=[G_1, \dots, G_n]$ - two frames of \mathcal{L}

Output: **True** iff $F \sqsubseteq G$ and **False** otherwise

- 1) Return **True** if and only if, for each $i \in [1, m]$:
 - a) If F_i is a role restriction, then there is $j \in [1, n]$ such that $G_j = F_i$
 - b) If F_i is a role value restriction of the form “ $R:C:\{V\}$ ”, or “ $R/1:C:\{V_1, \dots, V_k\}$ ”, or “ $R/2:C:\{V_1, \dots, V_k\}$ ”, then there is $j \in [1, n]$ such that $G_j = F_i$ or G_j is respectively of the form “ $R:C$ ”, or “ $R/1:C$ ”, or “ $R/2:C$ ”.
- 2) Return **False** otherwise. \square

Under the assumption that different constants denote different individuals, Algorithm FRAMESUBS has the following properties:

Proposition 4:

- i) Algorithm FRAMESUBS is $n \log n$ in the length of the longest argument.
- ii) Algorithm FRAMESUBS is sound, that is, if $\text{FRAMESUBS}[F, G]$ returns **True** then, for any interpretation I of \mathcal{L} , we have $F^I \sqsubseteq G^I$.
- iii) Algorithm FRAMESUBS is complete, that is, if for any interpretation I of \mathcal{L} , if $F^I \sqsubseteq G^I$, then $\text{FRAMESUBS}[F, G]$ returns **True**. \square

3.4 Extending slot subsumption for specific value domains

In the course of a prototype implementation of our algorithms, we experimented with an extended treatment of slot subsumption, to be mentioned informally here. By default, a conflict still occurs when trying to match two frames containing different values for the same slot name – except if special rules have been explicitly provided to indicate, for the value domain involved, how generalization and specialization should be executed.

Indeed, a value domain can have a lattice-like structure. For example, the value `techScience` could be a generalization of a number of disciplines, such as `chemistry` and `engineering`, whereas `design` and `engineering` could have `architecture` as specialization. This affects the results of both the `meet` and `join` operations on frames, and, as a consequence, lends some additional flexibility to classification. For example, let F_T be the frame for class `tech_student`, and F_B for the instance frame (for `Bertillon`), where

$F_B = [\text{area: engineering, fee: 45}]$

$F_T = [\text{area: techScience, fee:, name:, gender:, lodging:, monthly_pay:, restaurant:, room_phone:, scholarship:}]$

Then, F_B would now successfully match against F_T .

For properties of the binary relationship kind, special rules can restrict the value domain, which corresponds to an identifying attribute, to a more specialized entity class. So, if a new class `oil_industry` is introduced as a specialization of `company`, another new class `oil_worker` can be defined, as a specialization of `employee`, having the property `works/1:oil_industry` in its frame. In such cases, the subsumption process becomes recursive, since classifying an instance as an `oil_worker` involves verifying whether the instance pointed at by `works/1` can in fact be classified as an `oil_industry`.

Other different special rules can be likewise introduced for other value domains. At the cost of increased processing complexity, this feature clearly enhances the power of the search facilities which are the object of the next two sections.

4. Similarity

Besides trying to match an instance frame against a class frame, one may wish to find instances that match some *arbitrary* search frame. In some cases, one or more instances exactly matching the frame will be found, but, if none exists, an often quite useful option is to find instances that qualify as “good enough” approximations. In other words, instances that are *similar* to the one desired.

A modified version of the threefold closeness criterion used for classification leads to a mechanism to find – and compare – instances that do not necessarily satisfy all requirements formulated in a given arbitrary search frame. Matching an instance I with frame F_I against a search frame F_S can be understood as classifying F_I in an *ad-hoc* entity class. A fundamental difference is that classification involves traversing the lattice-like structure of predefined classes, until the process succeeds at some *depth*. For arbitrary search frames, as for classification, it makes sense to count the number of properties shared, as well as of value hits, but of course there is no question of depth.

On the other hand, when doing this kind of loosely restricted search with an arbitrary frame, all instances having at least one property in common may be of interest, even in the presence of conflicting values – although conflicting answers should be rated more poorly than the compatible ones. Thus, the criterion for lexicographic comparison adopted here also involves three items, counting, in this order of importance, how many value conflicts, properties shared, and value hits the frame of an instance has with respect to the given search frame. The number of conflicts receives a negative sign. An instance I will be taken as a better approximation than J if the closeness of I against the search frame is larger than that of J .

The response to a query formulated with a given search frame should list the instances retrieved, each one together with its triple so as to provide a measure of how closely it fits. An additional query parameter is most convenient: a *threshold*, also in the form of a triple, establishing a minimum acceptance requirement. As an example, if the search frame is `[area: law, fee: 45]`, and Bertillon's frame is `[area: engineering, fee: 45]`, he would be rated `[-1, 2, 1]`. Notice that this criterion is not the only possible one. In particular, the present criterion can be more finely tuned, for example by taking the difference between numerical values into consideration. So, an instance with `fee: 40` would rate better than one with `fee: 30`. The distance in hierarchy-

structured domain values could also be used for tuning. In this case, an instance with `area: management` would rate better than one with `area: chemistry`. Additionally, weights could be assigned to properties to rate their significance.

Since the search frame is arbitrarily formulated, it can be chosen to coincide with the frame of an existing instance. One would then be looking for instances similar to the given one. An especially important case arises from the notion of class *prototypes*. It has been argued [Lakoff & Johnson 1980] that the intuitive way that people use to judge whether an instance can be classified as belonging to a class is to measure its closeness to one (or to a few) prototypical representative(s) of the class. For example, a sparrow is often considered a prototype of the class bird, but a penguin never is. So, the similarity search described here can be used for this alternative method of classification, based on the instance frame of some indicated prototype of the intended class. The closeness criterion would then provide a measure of how central is a given instance to a class, by expressing its conformity to the prototype. Notice, however, that classifying solely on the basis of prototypes or examples may prove problematic in some cases, as extensively argued in [Medin 1989].

Another application of similarity [Barbosa & de Souza 2001] is to employ a known instance I_a , prototypical or not, in order to refer to an instance I_b , of which only one or a few non-identifying properties P_1, \dots, P_n have known values. We may ask, alluding this time to an application domain involving artists and countries: Who is the Brazilian Verdi (where `nationality` is an attribute)? or: Who is the Verdi of Brazil (assuming a relationship such as `born_in`)? This is a *correspondence* query, since it can be rephrased as: Who corresponds to Verdi in Brazil? Notice that the intuitive meaning of “ I_b corresponds to I_a ” is that I_b is similar to I_a with respect to the other properties. A correspondence query can thus be processed along the following steps:

- a) Retrieve the entire frame F_a of I_a (of Verdi, in the example).
- b) Replace in F_a the values of properties P_1, \dots, P_n by the values they have for I_b , obtaining a modified frame G_a .
- c) Perform a similarity search, as described before, with G_a .

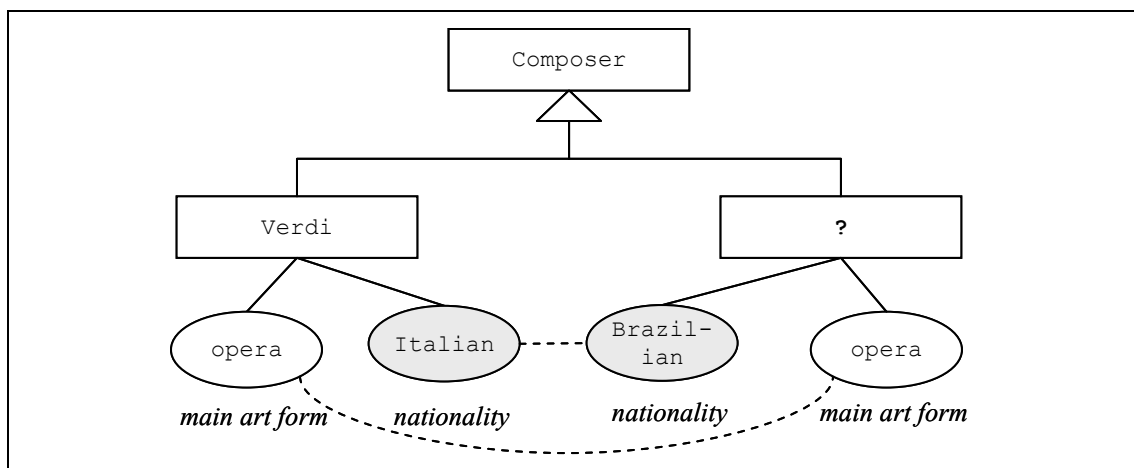


Figure 1. Illustration of the similarity search “Who’s the Brazilian Verdi?”.

In the example:

- a) Verdi's frame: $F_a = [\text{nationality: Italian, artistic skill: composer, main art form: opera, married_to/1: Margherita}]$
- b) Replace `nationality: Italian` by `nationality: Brazilian`
- c) Allow the user to inspect the search frame, and optionally mark some properties as irrelevant to the evaluation of closeness; in this case, `married_to/1` would be marked
- d) Search and find the instance frame with the highest closeness rate, in this case that of Carlos Gomes, even though, as one would anticipate, not all properties match:
 $F_a' = [\text{nationality: Brazilian, artistic skill: composer, main art form: opera, married_to/1: Adelina}]$

5. Analogy

We shall discuss two different ways to characterize analogy between two classes C_1 and C_2 . The first is through the *is-a hierarchy*: C_1 is analogous to C_2 if the two classes admit a non-trivial generalization C_m , i.e., if $C_m = C_1 \Delta C_2$ is not simply the universal class T . To be stricter, we may require that the number of common properties of C_1 and C_2 retained by C_m should be, either in absolute value or in terms of percentage, above a given threshold.

Thus, even if `fish` and `tiger` generalize to `animal`, it may be improper to call these classes analogous, whereas the generalization of `cat` and `tiger` to `feline` (and even more of `panther` and `tiger` to `wild feline`) would look far more significant. Notice that analogy through *is-a* is symmetric.

Analogy is a most useful notion. For example, if neither a specific instance I_1 of C_1 is available, nor even another instance J_1 of C_1 sufficiently similar to I_1 , then perhaps an instance I_2 of class C_2 , analogous to C_1 , is acceptable. `Puppy` and `goldfish` generalize to `pet`; a child might be persuaded to trade an instance of the former for a (less noisy) instance of the latter...

A non-symmetric form of analogy can be established via *is-like connections*. Intuitively, by declaring that a class C_1 is-like another class C_2 , denoted $C_1 \rightarrow C_2$, one tries to characterize the usually more abstract or generally not so well-known class C_1 , called

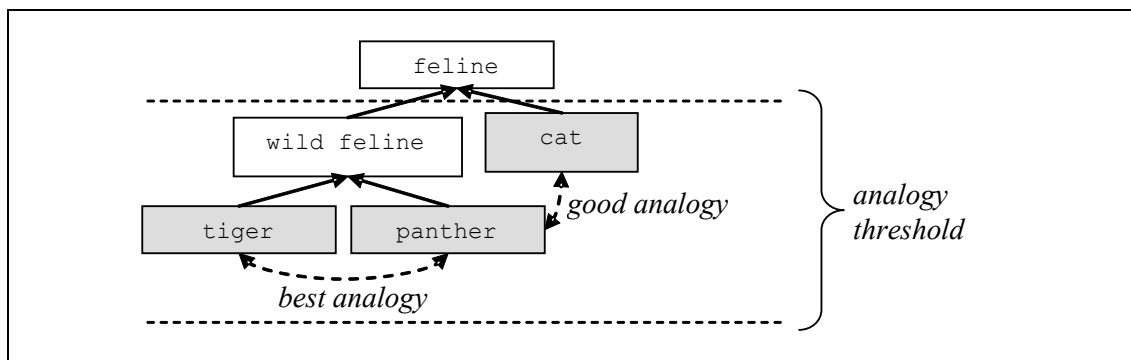


Figure 2. Illustration of symmetric analogies in an is-a hierarchy.

SBBD 2007
XXII Simpósio Brasileiro de Banco de Dados

the *target* class, in terms of the more concrete or better known class C_2 , called the *source* class. To be effective, the declaration must include the specification of *property mappings* $P_1 \rightarrow P_2$, to characterize those properties of C_1 that somehow resemble properties of C_2 . It is by this partial resemblance that the *is-like* connection is justified, allowing us to say that C_1 is, to some extent, analogous to C_2 . For example, the declaration below indicates two attributes and one relationship of the class `company` that are reminiscent of properties of the class `person`:

```
(company→person)[acronym→name, headquarters→address,  
affiliated_to/1→son_of/1])
```

Talking about a company as if it were a person constitutes a figure of speech named *metaphor* [Lakoff & Johnson 1980, Way 1994]. In fact, it has been argued that companies (or organizations, more generally) can be explained in terms of several different metaphors, each one providing the basis for quite distinct theories of organization [Morgan 1998].

It is a known fact that users often tend to formulate queries about a target class employing the terminology of a source class. For example, they may ask the “address” of a `company`, which a friendly interface, aware of the *is-like* connection, would automatically replace by `headquarters` before processing the query.

Correspondence queries, mentioned in the previous section as an application of similarity, can be extended through the use of analogy. Take, for example, classes `employee`, `sale`, `player`, and `game`. Assume that the following binary relationships have been specified:

```
achieved(employee, sale)                won(player, game)
```

Assume further that an *is-like* connection (between two classes) `employee→player` has been explicitly declared, with the attached property mapping `achieved→won`. For consistency, another connection should be inferred as a consequence, namely `sale→game`, as depicted in Figure 3.

Now, if the two facts below happen to be true:

```
achieved(John Smith, sale#123)          won(Federer, someGame)
```

a query such as “Who is the Federer of sale#123?” would qualify as a correspondence query, yielding, John Smith as the result.

Several other applications of analogy come easily to mind, two of which will be briefly outlined. In the course of the conceptual design phase of an application domain, if a class C_2 is already known (in that domain or another), then, after the connection $C_1 \rightarrow C_2$ is declared, an interactive design support tool could help the designer to specify properties of C_1 by means of prompts of the form “Does C_1 possess some property analogous to property P_i of C_2 ?”.

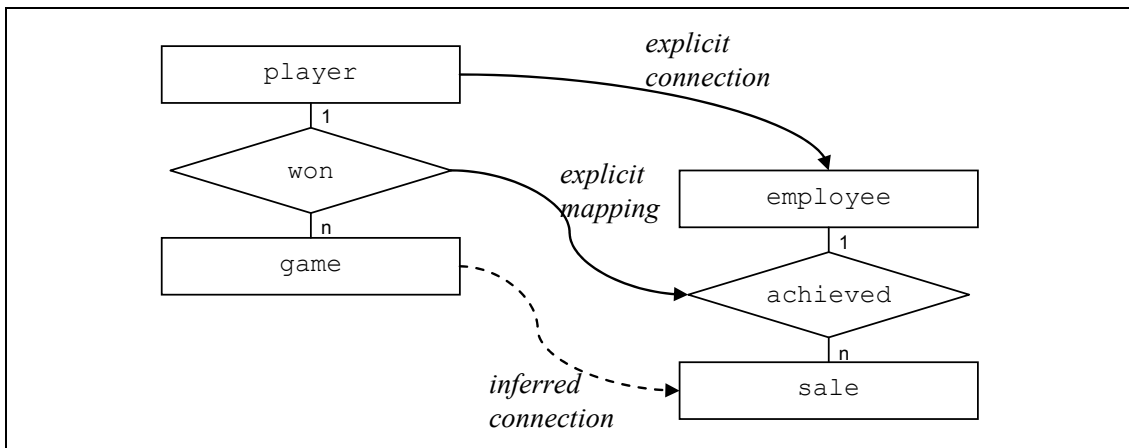


Figure 3. Analogy mapping between binary relationships.

The other application aims at reducing the computational complexity of tasks involving classes with a large number of properties, which may be overwhelming in some cases [French 1997]. Suppose one such class C has been declared as the target of different source classes C_1, C_2, \dots, C_n , the respective property mappings having been specified for each of the *is-like* connections. Then, as a task is initiated involving class C , it becomes possible, as a preliminary step, to restrict the set of properties of C to be considered by asserting: take C as $\{C_{i1}, C_{i2}, \dots, C_{in}\}$, for i_j in $[1, n]$. The intended effect would be to consider only the properties of C figuring in the respective property mappings. This is tantamount to asserting on which aspects of C one wishes to focus, thereby defining a limited *perspective* under which C is to be viewed.

6. Conclusion

Similarity and analogy are powerful notions, whose practical adoption contributes to a semantically richer design of information systems, and expands the range of queries that can be formulated over their application domains. We have been investigating the use of similarity and analogy with the help of prototype tools, implemented with Logic Programming software. Their use via interfaces with existing database management systems are part of the future goals of our project. A companion paper [Breitman et al. 2007] discusses analogy and metaphor in the context of conceptual database design.

References

- Baader, F. and Nutt, W. (2003) "Basic description logics". In *The Description Logic Handbook: Theory, Implementation and Applications*. Baader, F.; Calvanese, D.; McGuinness, D.L.; Nardi, D.; Patel-Schneider, P.F. (eds.). Cambridge University Press.
- Barbosa, S. D. J. and de Souza, C. S. (2001) "Extending software through metaphors and metonymies". In *Knowledge-Based Systems*, 14.
- Batini, C., Ceri, S. and Navathe, S. (1992) *Conceptual Design – an Entity-Relationship Approach*. Benjamin Cummings.
- Brachman, R. and Levesque, H. (2004) *Knowledge Representation and Reasoning*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufman.
- Breitman, K. K., Barbosa, S. D. J., Furtado, A. L. and Casanova, M. A. (2007) "Conceptual modeling by analogy and metaphor". (Accepted CIKM 2007).
- Fankhauser, P., Kracker, M., and Neuhold, E. J. 1991. Semantic vs. structural resemblance of classes. SIGMOD Rec. 20, 4 (Dec. 1991), 59-63.

SBBD 2007
XXII Simpósio Brasileiro de Banco de Dados

- Franconi, E. (2002) "Structural Description Logics: FL⁻". In *Description Logics Course*. Available at: <http://www.inf.unibz.it/~franconi/dl/course/slides/struct-DL/flminus.pdf>
- French, R. (1997) "When coffee cups are like old elephants - or why Representation modules don't make sense". In *Proc. International Conference on New Trends in Cognitive Science*.
- Gentner, D. and Markman, A. B. (1997) "Structure mapping in analogy and similarity". *American Psychologist* 52.
- Holyoak, K. J. and Thagard, P. (1996) *Mental Leaps: Analogy in Creative Thought*. MIT Press.
- Lakoff, G. and Johnson, M. (1980) *Metaphors We Live By*. University of Chicago Press.
- MacLane, S. and Birkhoff, G. (1967) *Algebra*. MacMillan.
- Medin, D. L. (1989) "Concepts and conceptual structure". *American Psychologist* 44.
- Morgan, G. (1998) *Images of organization - Executive edition*. Sage Publications.
- Nardi, D.; Brachman, R.J. (2003) "An introduction to description logics". In: Baader, F.; Calvanese, D.; McGuinness, D.L.; Nardi, D.; Patel-Schneider, P.F. (Eds) *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge, UK.
- Schank, R. C. and Colby, K. M. (1973) *Computer models of thought and language*. W. H. Freeman & Co
- Way, E. C. (1994) *Knowledge Representation and Metaphor*. Intellect Books.

Annex - A Minimal Review of Attributive Languages

Description logic (DL) models the application domain by defining the relevant concepts of the domain and then using these concepts to specify properties of objects and individuals occurring in the domain (Baader and Nutt 2003). An *attributive language* \mathcal{L} is characterized by an *alphabet* consisting of a set of *atomic concepts*, a set of *atomic roles*, and the special symbols \top and \perp , respectively called the *universal concept* and the *bottom concept*. The set of concept descriptions of \mathcal{L} is inductively defined as follows:

- Any atomic concept and the universal and bottom concepts are concept descriptions.
- If C and D are concept descriptions, and R is an atomic role, then the following expressions are concept descriptions

$\neg C$	(negation)	R^{-}	(the inverse of R)
$C \sqcap D$	(intersection)	$C \sqcup D$	(union)
$\forall R.C$	(value restriction)	$\exists R.C$	(full existential quantification)
$(\geq n R)$	(at-least restriction)	$(\leq n R)$	(at-most restriction)

An *interpretation* I for \mathcal{L} consists of a nonempty set Δ^I , the *domain* of I , whose elements are called *individuals*, and an *interpretation function* such that:

- $\top^I = \Delta^I$ and $\perp^I = \emptyset$
- For every atomic concept A of \mathcal{L} , the function assigns a set $A^I \subseteq \Delta^I$
- For every atomic role R of \mathcal{L} , the function assigns a binary relation $R^I \subseteq \Delta^I \times \Delta^I$

The interpretation function is extended to concept descriptions of \mathcal{L} as follows:

- $(\neg C)^I$ denotes the complement of C^I with respect to the domain
- $(R^{-})^I$ denotes the inverse of R^I
- $(C \sqcap D)^I$ denotes the intersection of C^I and D^I
- $(C \sqcup D)^I$ denotes the union of C^I and D^I
- $(\forall R.C)^I$ denotes the set of individuals that R relates only to individuals in C^I , if any
- $(\exists R.C)^I$ denotes the set of individuals that R relates to some individual in C^I
- $(\geq n R)^I$ denotes the set of individuals that R relates to at least n individuals
- $(\leq n R)^I$ denotes the set of individuals that R relates to at most n individuals

A *terminological axiom* (written) in \mathcal{L} or, simply, an *axiom*, is an expression of the form $C \sqsubseteq D$, called an *inclusion*, or of the form $C \equiv D$, called an *equality*, where C and D are concept descriptions in \mathcal{L} . Let I be an interpretation for \mathcal{L} . Then, I satisfies $C \sqsubseteq D$ iff $C^I \subseteq D^I$, and I satisfies $C \equiv D$ iff $C^I = D^I$.