Specialization by Restriction and Schema Derivations

Anastasia Analyti[1], Panos Constantopoulos[1,2], Nicolas Spyratos[3] *

[1] Institute of Computer Science, Foundation for Research and Technology-Hellas, Greece
[2] Department of Computer Science, University of Crete, Greece
[3] Laboratoire de Recherche en Informatique, Universite de Paris-Sud, France
E-mail: analyti@ics.forth.gr, panos@ics.forth.gr, spyratos@lri.fr

Abstract

Specialization and inheritance are well-known concepts in the area of object-oriented modelling and knowledge representation. However, certain aspects of these concepts lack formal foundations. In particular, when properties of different classes are semantically related, several *different* semantics are possible for the inherited properties, and a choice is necessary. Conventional systems impose an a priori solution that supports only one of the possible semantics of inheritance.

In this paper, we present constructs that allow to differentiate between the possible semantics of inheritance, in a formal and sound way. Our approach is based on a structured view of the real world and a model for its representation. By necessity, the model only partly represents the real world. Thus, reasoning in the model is done with reference to the real world.

We introduce *restriction isa*, a form of specialization that represents property restriction, and demonstrate that it can be a useful conceptual modelling mechanism. We employ restriction isa to formally define property inheritance.

Reasoning in our model is done through a number of inference rules that reflect real world constraints. These rules allow for sound derivations both at instance and schema level. At the schema level, in particular, these rules allow to relate inherited properties to other properties through restriction-isa and isa relations. Such relations not only give useful information about the inherited properties but also refine the value domains of these properties.

Keywords: Property inheritance, semantics, inference rules, schema derivations, conceptual modelling, object-oriented modelling.

1 Introduction

Specialization and inheritance are well established mechanisms in object-oriented modelling and conceptual modelling [9, 18, 21, 24]. Inheritance typically takes place from classes to

*Research conducted while this author was visiting with the Institute of Computer Science, Foundation for Research and Technology-Hellas.

instances: a class is a container of information common to its instances and each instance inherits a number of properties by class membership [22].

Specialization hierarchies, on the other hand, allow to organize classes in a way that a subclass contains only declarations of properties that do not appear in its superclasses. This implies that a class declaration may not contain explicitly all information common to the class instances and specialization may be consulted in order to obtain a complete set of properties for the class instances. In a sense we can say that a class inherits all properties that appear in its superclasses. This is because each instance of a class is also an instance of its superclasses.

This approach is unambiguous in the case that (i) the local properties of the subclass are not related semantically to the properties of its superclasses, and (ii) the properties inherited by the subclass from its superclasses are not semantically related among themselves. As an example of (i), assume that the class *Person* has the property *address* and its subclass *Employee* has the property *salary*. As these two properties are not semantically related, inheritance of *address* by the subclass *Employee* presents no problem.

However, in cases other than (i) and (ii) above, inheritance may pose problems, as several *different* semantics are possible for the inherited properties. For example, assume that the class *Vehicle factory* refers to factories producing vehicles, and that *Vehicle factory* has a subclass *Car factory* that refers to factories producing cars. Moreover, assume that *Vehicle factory* has a property *produces* that takes values in class *Vehicle*, whereas *Car factory* has a property *produces* that takes values in class *Car*. Clearly, these two properties are related semantically, and there might be some confusion as to the interpretation of class *Car factory*.

Indeed, an instance o of *Car factory* inherits the property *produces* of *Car factory* by being an instance of the class. However, as o is also an instance of *Vehicle factory*, it also inherits the property *produces* of *Vehicle factory*. This leads to two possible interpretations of *Car factory*:

Option 1: A car factory produces cars but can also produce vehicles other than cars.

Option 2: A car factory produces only cars.

Option 1 represents a "liberal" interpretation of *Car factory*, as it allows instances of *Car factory* that produce vehicles other than cars. Option 2 represents a "conservative" interpretation, as it forces instances of *Car factory* to produce cars only. In this case, we can say that the property *produces* of *Car factory* "refines" the property *produces* inherited from *Vehicle factory*. "Refinement" here is interpreted as *value restriction*.

Note that both options 1 and 2 are consistent with the so-called "strict inheritance". Namely, a class definition can only add new properties or restrict the value domain of the property inherited from a superclass.

Now the question is how to indicate to the system which of the two options gives the desired semantics for *Car factory*. Conventional data models, such as [1, 2, 3, 8, 14, 16, 26, 27, 28] provide specialization mechanisms that support only one of the two options (see also section 6). We believe that both options should be supported, as the appropriate option depends on the desired class semantics. Moreover, we believe that an a priori choice of one or the other option can lead to problems. For example, suppose that Option 1 is imposed by the system as the semantics of *Car factory*, and that it is Option 2 that really represents the intended semantics. Such a situation can lead to problems for two reasons. First, because it allows the insertion of invalid information (insertion of a particular car factory that produces a vehicle other than car). Second, because it may lead the casual user who is browsing through the schema to a false interpretation of *Car factory*. Similar problems arise if Option 2 is imposed by the system and it is Option 1 that really represents the intended semantics.

In this paper, we present a model that differentiates class semantics with respect to property inheritance. We introduce the *restriction isa* relation (denoted *Risa*), a form of specialization that represents property restriction. We demonstrate that restriction isa can be a powerful and useful conceptual modelling mechanism. We further employ *Risa* in formalizing property inheritance by specialization, a notion that generally lacks formal foundations and is mostly defined and used empirically.

To get a feeling of how *Risa* can differentiate between the possible semantics of inheritance, refer to Figure 1, where thin arrows represent properties of classes. Properties can be related through the usual isa relation or through the restriction isa (these two relations are denoted by different types of arrows as shown in the figure). The class *Painting collector* has a property *collects* that is related to the property *collects* of its superclass *Art collector* by a usual isa relation (declared by the user). This declaration indicates that the semantics of *Painting collector* is as in *Option* 1. In other words, a painting collector collects paintings

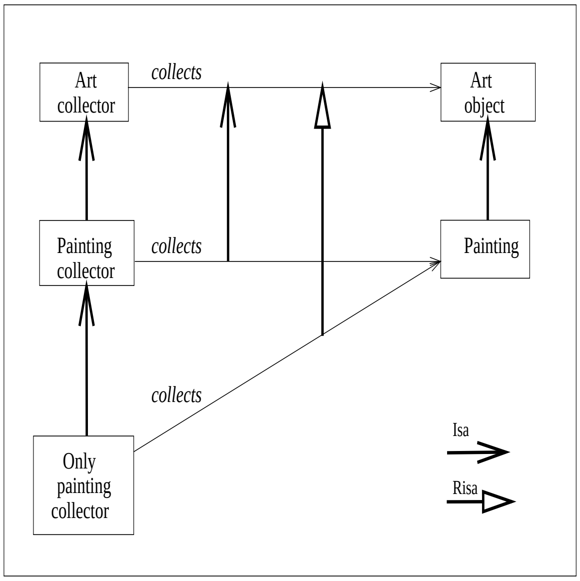

Figure 1: Example of multiple semantics of property inheritance

but can also collect art objects other than paintings. Therefore, the value domain of the property inherited by *Painting collector* from *Art collector* should be *Art object.*

On the other hand, the class *Only painting collector* has a property *collects* that is related to the property *collects* of *Art collector* by the restriction isa relation (also declared by the user). This declaration indicates that the semantics of *Only painting collector* is as in *Option* 2. In other words, an only painting collector collects only paintings. Therefore, the value domain of the property inherited by *Only painting collector* from *Art collector* should be *Painting.*

Reasoning in our model is done through a number of inference rules that allow for derivations both at instance and schema level. At the schema level, in particular, these rules allow to relate inherited properties to other properties through isa and restriction-isa relations. Such relations not only give useful information about the inherited properties but also refine the value domains of these properties.

For example, referring to Figure 2, consider the class *Rich painting collector* which is subclass of both *Painting collector* and *Rich art collector.* The property *collects* of *Rich art collector* is restriction subclass of the property *collects* of *Art collector* (indicating that all art objects that are collected by rich art collectors are expensive). Note that the class *Rich painting collector* does not have any explicitly declared property. However, it inherits the properties of its superclasses. For example, *Rich painting collector* inherits the property

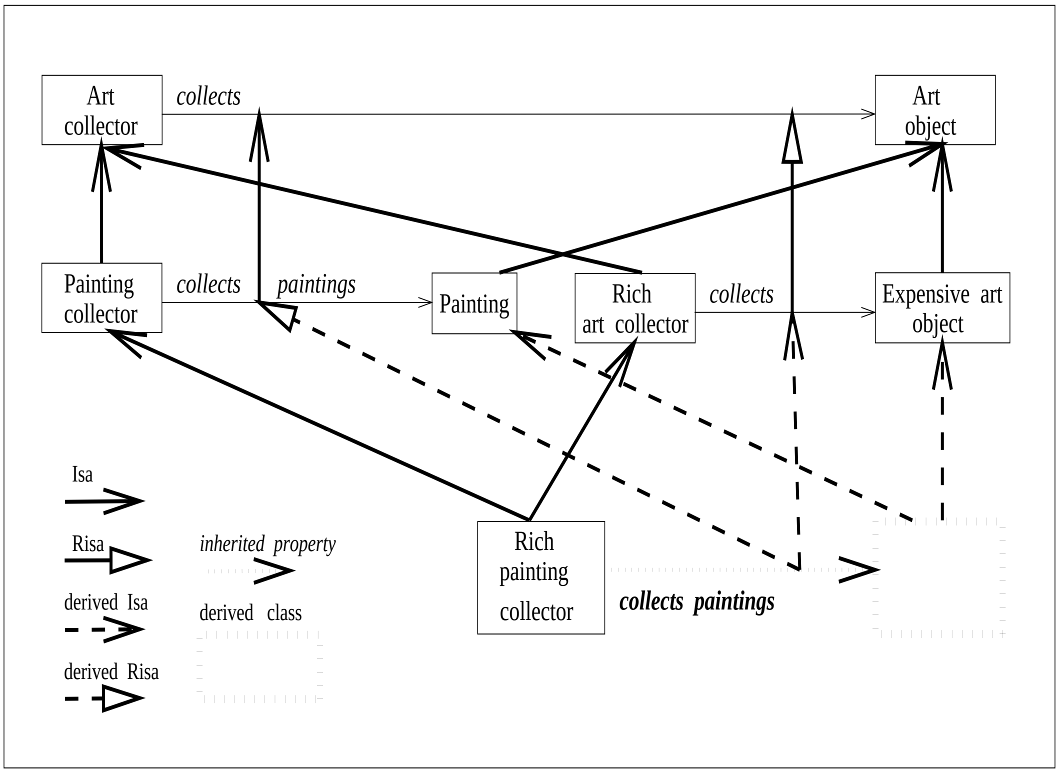

Figure 2: Example of inherited property

collects paintings of *Painting collector*. From inference rules, we derive that the inherited property is a restriction subclass of the property *collects paintings* of *Painting collector* and a subclass of the property *collects* of *Rich art collector*. Additionally, the value domain of the inherited property is subclass of both *Painting* and *Expensive art object*. This indicates that all the paintings that rich painting collectors collect are expensive art objects.

Conventional object-oriented systems will consider the value domain of the inherited property to be *Painting*. Thus, they do not fully exploit the information provided by the schema. On the other hand, as we shall see, our approach considers the value domain of the inherited property to be a subclass of both *Painting* and *Expensive art object*.

The restriction isa relation is used for derivations at both the schema and instance levels. In our previous example, the restriction isa relation was used for the derivation of a new property of the class *Rich painting collector*. At the schema level, information is derived exploiting isa and restriction isa relations. At the instance level, information is derived based on isa, restriction isa, and instance-of relations.

In our work we adopt an open-world approach that differentiates between information about an object in the real world and information available in the model. That is, information in the model may not be complete. Soundness of the inference rules in the model is proved with respect to relations that hold in the real world.

The information representation framework adopted in our work follows quite closely the structural part of Telos [23, 19]. Telos is a knowledge representation language that attempts to integrate deductive and object-oriented features. However, in the definition of the semantics of Telos, property inheritance by specialization is not treated in depth.

In our model, inherited properties are considered to be "first-class", i.e., they have distinct identity. This is important as inherited properties *may not* have all the characteristics of the original properties. For example, consider the class *Parent* with the property *has-child*. One of the characteristics of this property is that it represents a 2:M relationship (a parent has 0,1, or more children, whereas a child always has 2 parents). Consider now the class *Mother* which is subclass of *Parent*. The property *has-child* is inherited by *Mother*. However, the inherited property should not be assumed to be 2:M (as a child has only one mother).

Systems that do not differentiate between the original and the inherited property, implicitly assume that the inherited property has all the characteristics of the original property. In our example, they will implicitly assume that the property inherited by $Mother$ is also 2:M, obviously a false assumption.

In our model, all information concerning inherited properties is derived through the inference rules alone. So, the property *has-child* inherited by *Mother* will not be automatically assumed to be 2:M.

In summary, we can say that the contributions of the present work are the following:

1. Support for different possible semantics of inheritance.

2. Automatic refinement of the value domain of the inherited properties, based on a set of inference rules.

3. Clear separation between the real world, where information is considered to be complete, and the model, where information is incomplete.

4. Formal justification of the inference rules of the model, based on reasoning over real world objects and their relations.

5. Schema derivations through the restriction isa relation.

The operational context of our work is the Semantic Index System (SIS) [10, 11, 12, 13]. Information in the SIS is represented by an object-oriented semantic network, where nodes and links are uniformly treated, and multiple instantiation, multiple specialization, and virtually unlimited instantiation levels are supported. An interval-based time primitive is included along with appropriate temporal operators. Specific deductive rules and static integrity constraints are also supported through special semantic structures.

The rest of the paper is organized as follows: Section 2 describes our view of the real world. Section 3 presents the basic constructs of our model for representing the real world. In section 4, property specialization by restriction is defined and its basic properties are explored. A set of inference rules related to restriction isa are presented. Section 5 describes our view on property inheritance and defines property inheritance based on the restriction isa relation. A set of inference rules related to property inheritance are presented. In section 6, we review related work. Section 7 contains concluding remarks.

All proofs are given in the Appendix.

2 The "real world"

In this section, we present our view of the real world. First, we describe real world objects in terms of intention and extension. Then, based on this description, we relate real world objects through the *instantiation* and *specialization* relations[1].

Real world objects, or *real objects* for short, are distinguished with respect to their nature in: *real individuals* and *real arrows*.

- **Real individuals**: These are concrete or abstract entities of independent existence. Concrete entities are things (simple or composite), events, processes. Abstract entities refer collectively to entities which are considered similar in some respect.

  Examples of real individuals are: (i) the concrete entities MY FATHER and MY CAR, (ii) the abstract entity CAR, that refers collectively to all cars (including MY CAR), and (iii) the abstract entity VEHICLE TYPE, that refers collectively to all vehicle types (including CAR).

[1]Throughout the paper, we use the term *relation* with its mathematical meaning.

- **Real arrows**: These are concrete or abstract properties of real objects. More precisely, a real arrow represents a property of an object O with value a real object O'. The real objects O, O' are called the $from$ and to object of the real arrow, respectively.

  Real arrows, in contrast to real individuals, do not exist independently: their existence presumes the existence of the $from$ and to objects.

  Examples of concrete properties are: (i) the property PAID-FOR from MY FATHER to MY CAR (meaning that my father paid for my car), and (ii) the property REPAIRED from MY FATHER to MY CAR (meaning that my father repaired my car).

  An example of abstract property is the property PRODUCES from CAR-COMPANY to CAR, which refers collectively to all concrete properties from car companies to the cars that they produce.

  The $from$ object of a real arrow A is denoted by $from(A)$ and the to object by $to(A)$.

The distinction of real objects to real individuals and real arrows is based on their nature. Real objects are also distinguished with respect to their concreteness, in *real tokens* and *real classes*.

- **Real Tokens:** These are concrete real objects.

  For example, the concrete real objects MY FATHER and MY CAR are real tokens. We have already seen that they are also real individuals. Therefore, we call them *real individual tokens*.

  The concrete real objects PAID-FOR from MY FATHER to MY CAR and REPAIRED from MY FATHER to MY CAR are real tokens. As they are also real arrows, we call them *real arrow tokens*.

- **Real Classes:** These are abstract real objects, in the sense that they refer collectively to a set of real objects that are considered similar in some respect.

  For example, the abstract object CAR is a real class. As it is also a real individual, we call it a *real individual class*.

  The abstract object PRODUCES from CAR-COMPANY to CAR is a real class. As it is also a real arrow, we call it a *real arrow class*.

The distinction of real objects with respect to their nature and with respect to their concreteness is illustrated in the left part of Figure 3.

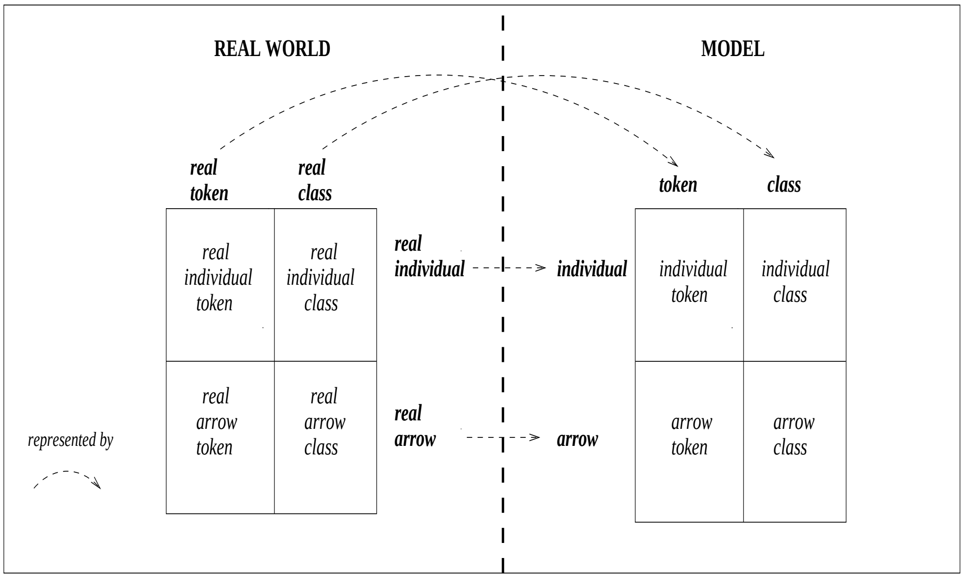

Figure 3: Real world and its representation in the model

We assume that a real object is defined by a set of constraints, called the *intention* of the object. For a real class, the intention determines the set of real objects to which the real class refers collectively. We call this set the *extension* of the real class. The extension of a real individual class is a set of real individuals. The extension of a real arrow class from a real class O to a real class O' is a set of real arrows from objects in the extension of O to objects in the extension of O'. Thus, the extension of a real arrow class defines a relation from the extension of O to the extension of O'.

For example, the intention of the real individual token MY CAR is the set of constraints that define my car. The intention of the real individual class CAR is the set of constraints that define a car, and the extension of CAR is the set of all cars. The extension of the real arrow class PRODUCES from CAR-COMPANY to CAR is the set of all real arrow tokens *from* car companies *to* the cars they produce.

For the purposes of this paper, we make the following assumptions:

- *Individual equality assumption*

  Any two real individual classes with identical extensions have identical intentions.

- *Arrow equality assumption*

  Any two real arrow classes with the same *from* and *to* objects and identical extensions have identical intentions.

Note that if a real individual is viewed as a real arrow from the individual to itself then the arrow equality assumption implies the individual equality assumption.

Real objects can be related to real classes through the *instance of* relation.

Definition 2.1 IN Relation

If a real object O belongs to the extension of a real object C then we say that O is an *instance of* C, and we denote it by $\mathrm{IN}(O, C)$. A real object can be an instance of zero, one, or more real classes.

It follows from the definition of extension of real classes that instances of real individual classes are real individuals and instances of real arrow classes are real arrows. Additionally, if X and A are real arrows such that X is an instance of A, then $from(X)$ is an instance of $from(A)$ and $to(X)$ is an instance of $to(A)$.

From the earlier definition of intention and extension of a real class, it follows that the stronger the intention the smaller the extension. This property of real classes is expressed in the following definition of the ISA relation.

Definition 2.2 ISA Relation

Let C, C' be two real individual classes. If the extension of C is a subset of the extension of C' then we say that C is a *subclass* of C', and we denote it by $\mathrm{ISA}(C, C')$.

Let A, A' be two real arrow classes. If $from(A)$ is a subclass of $from(A')$, $to(A)$ is a subclass of $to(A')$, and the extension of A is a subset of the extension of A' then we say that A is a *subclass* of A', and we denote it by $\mathrm{ISA}(A, A')$.

A real class can be subclass of zero, one, or more real classes.

The ISA relation satisfies the following properties (see proofs in the Appendix):

ISA FACTS

Let O be any real object and C, C', C'' be any real classes.

Fact 1: If $\mathrm{ISA}(C, C')$ and $\mathrm{IN}(O, C)$ then $\mathrm{IN}(O, C')$.

Fact 2: *Reflexivity:* $\mathrm{ISA}(C, C)$.

Fact 3: *Transitivity:* If $\mathrm{ISA}(C, C')$ and $\mathrm{ISA}(C', C'')$ then $\mathrm{ISA}(C, C'')$.

Fact 4: *Antisymmetry:* If $\mathrm{ISA}(C, C')$ and $\mathrm{ISA}(C', C)$ then C coincides with C'.

From the reflexivity, transitivity, and antisymmetry properties of ISA, it follows that the ISA relation defines a partial order over real classes.

3 Basic concepts of the model

In the previous section, we described real world objects and their *instantiation* and *specialization* relations. In this section, we define their counterparts in our model.

A real object is represented in our model by what we call a *model object*, or simply *object*. As we have explained, a real object is identified by its intention. An object in the model, however, is identified just by a unique identifier (for example an integer). This is because the intention of a real object cannot always be expressed in the model.

An object in the model is called *individual* or *arrow*, if it represents a real individual or a real arrow, respectively. An object is called *token* or *class*, if it represents a real token or a real class, respectively. This is illustrated in Figure 3.

Each individual has a *name* which is globally unique. Each arrow has a *label* which is unique within its *from* object (but not necessarily globally unique).

We use capitalized names for real objects and the same names in small letters for their corresponding objects in the model. For example, we use MY CAR for the real individual and *my car* for the corresponding individual in the model. Similarly, we use PRODUCES for the real arrow and *produces* for the corresponding arrow in the model.

3.1 Instantiation

Any two objects in the model can be related through a relation, denoted by In. The pairs of objects that are related through this relation are either declared by the user or derived by the system. The declaration of pairs by the user is done under the following assumption: If $In(o, c)$ is declared by the user then $\mathrm{IN}(O, C)$ holds, for any objects o, c representing the real objects O, C, respectively.

If $In(o, c)$ holds then we say that o is an *instance of* c, or c is a *class of* o. The *extension* of a class c is defined to be the set of instances of c.

Let x and a be two arrows such that $In(x, a)$. Let X be the real arrow represented by x and A be the real arrow represented by a. It follows that X is an instance of A. This implies that $from(X)$ is an instance of $from(A)$ and $to(X)$ is an instance of $to(A)$.

Therefore, $from(x)$ should be an instance of $from(a)$ and $to(x)$ an instance of $to(a)$. Thus, an arrow x with $from$ object o can be instance only of arrow classes whose $from$ object is a class of o. Intuitively, arrow classes (resp. arrow tokens) whose $from$ object is a class c correspond to instance (resp. class) variables[2] of c.

Note that the extension of an object c is defined through the In relation, so it carries less information than the extension of the real object that c represents. Indeed, if a real object O is an instance of a real object C then we may have one of the following cases: (i) O and/or C are not represented in the model, (ii) O and C are represented by objects o, c, but $\mathrm{IN}(O, C)$ is not represented in the model, i.e., $In(o, c)$ does not hold, and (iii) O and C are represented by objects o, c, and $In(o, c)$ holds. Because of cases (i) and (ii), the extension of an object may carry less information than the extension of the real object it represents. For example, the extension of the object car carries less information than the extension of the real object CAR, because only some of the real world cars are represented in the model.

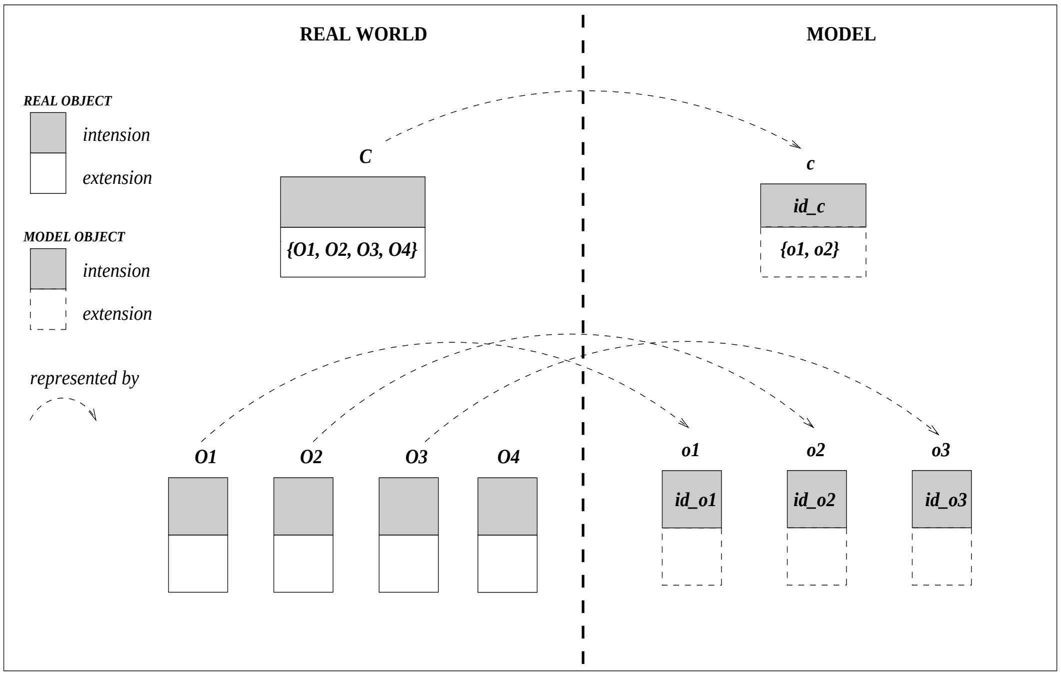

Figure 4: Real classes and their representation in the model

[2]In the object-oriented terminology [18], an *instance variable* (also called *member attribute*) of a class c is a property declaration for which instances of c may carry a specific value. A *class variable* (also called *type attribute*) of c corresponds to a property of c as a whole, e.g., the average number of instances of c.

Figure 4 illustrates several concepts related to instantiation. In the figure, the real class C has instances $O1$, $O2$, $O3$, and $O4$. Real objects C, $O1$, $O2$, $O3$ are represented in the model by the objects c, $o1$, $o2$, $o3$, whereas the real object $O4$ is not represented in the model. Additionally, although $O3$ is an instance of C, this information is not represented in the model.

The In relation between individuals appears in many data models. Our view on instantiation was inspired by Telos [23] where the In relation is applicable not only to individuals but also to arrows. An overview of manifestations of instantiation in several systems is given in [22].

3.2 Specialization

Any two classes in the model can be related through a relation, denoted by Isa. The pairs of classes that are related through this relation are either declared by the user or derived by the system. The declaration of pairs by the user is done under the following assumption: If $Isa(c, c')$ is declared by the user then $\text{ISA}(C, C')$ holds, for any classes c, c' representing the real classes C, C', respectively.

If $Isa(c, c')$ holds then we say that c is a *subclass* of c', or that c is a *specialization* of c', or that c' is a *superclass* of c.

It is important to note that two objects may not be related through the Isa relation, although the real objects they represent are related through the ISA relation. This is because knowledge of the real world is incomplete.

3.3 Inference Rules

As we mentioned earlier, the In and Isa relations are either declared by the user or derived by the system.

Derivations are performed using certain inference rules that reflect properties of the real world relations IN and ISA. For example, as we have already indicated, only a subset of the extension of a real class is represented in the model. Therefore, although the user may have declared $Isa(c, c')$, it may happen that the user-declared extension of c is *not* a subset of the user-declared extension of c'. In other words, there may be an object o which is explicitly declared as an instance of c but not as an instance of c'. In this case, the system will infer that o is an instance of c', that is, $In(o, c')$ will be a derived relation.

The soundness of our derivation rules is proved based on the relations that hold in the real world. This is because information in the real world is complete, whereas information in the model is incomplete. For example, in the model, if the extension of a class c is a subset of the extension of a class c' we *cannot* derive that $Isa(c, c')$ holds. In contrast, in the real world, if the extension of a real class C is a subset of the extension of a real class C' then $\text{ISA}(C, C')$ holds, and from this we can derive that $Isa(c, c')$ holds.

There are several rules for deriving new In and Isa relations based on existing ones. The soundness of these rules is proved in the Appendix.

Isa RULES

Let o be any object and c, c', c'' be any classes.

Rule 1: If $In(o, c)$ and $Isa(c, c')$ then $In(o, c')$.

Rule 2: *Reflexivity:* $Isa(c, c)$.

Rule 3: *Transitivity:* If $Isa(c, c')$ and $Isa(c', c'')$ then $Isa(c, c'')$.

Note that there is no rule that corresponds to ISA Fact 4 (antisymmetry). This is because, antisymmetry is reflected in the model through a constraint and not through an inference rule, as we explain next.

3.4 Semantic Constraints

As we have seen, In and Isa represent relations in the real world that satisfy certain constraints. In the model, some of these constraints are translated into inference rules and others into semantic constraints. The semantic constraints guarantee the validity of the information in the model.

CONSTRAINTS

1. *Arrow instantiation constraint*

   Let x, a be arrows. If $In(x, a)$ then the *from* and *to* objects of x must be instances of the *from* and *to* objects of a, respectively.

   This constraint is illustrated in Figure 5 (a).

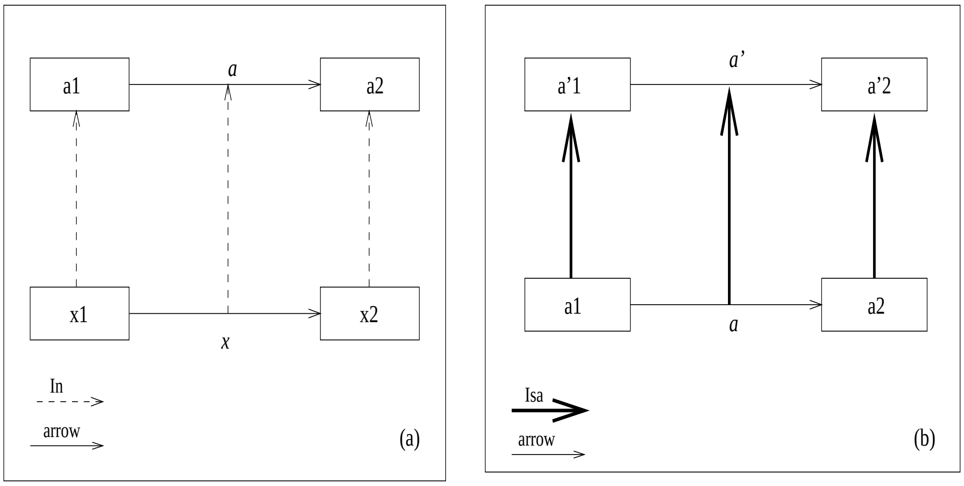

Figure 5: (a) Arrow instantiation constraint, (b) Arrow specialization constraint

2. *Arrow specialization constraint*

   Let a, a' be arrows. If $Isa(a, a')$ then the *from* and *to* objects of a must be subclasses of the *from* and *to* objects of a', respectively.

   This constraint is illustrated in Figure 5 (b).

3. *Isa antisymmetry constraint*

   Let c, c' be classes. If $Isa(c, c')$ and $Isa(c', c)$ then c and c' must coincide.

The above constraints are easily justified from the ISA and IN relations in the real world. Though suitable forms of these constraints could be used as inference rules, we have decided to use them as constraints for checking the validity of the explicitly-declared information.

For example, the *Isa antisymmetry constraint* is justified by the ISA antisymmetry property. The inference rule corresponding to this constraint would have been the following: If c is a subclass of c' and c' is a subclass of c then (infer that) c and c' coincide. Obviously, to infer that two explicitly declared classes coincide may lead to wrong conclusions. Thus, in this case, we thought it more appropriate, rather than inferring that the two classes coincide, to indicate the problem to the user.

3.5 An example

In this subsection, we illustrate the In and Isa relations through a more detailed example.

Referring to Figure 6, recall that although arrow labels are unique within their $from$ objects, they may not be globally unique. For example, *engine* is the label of two different

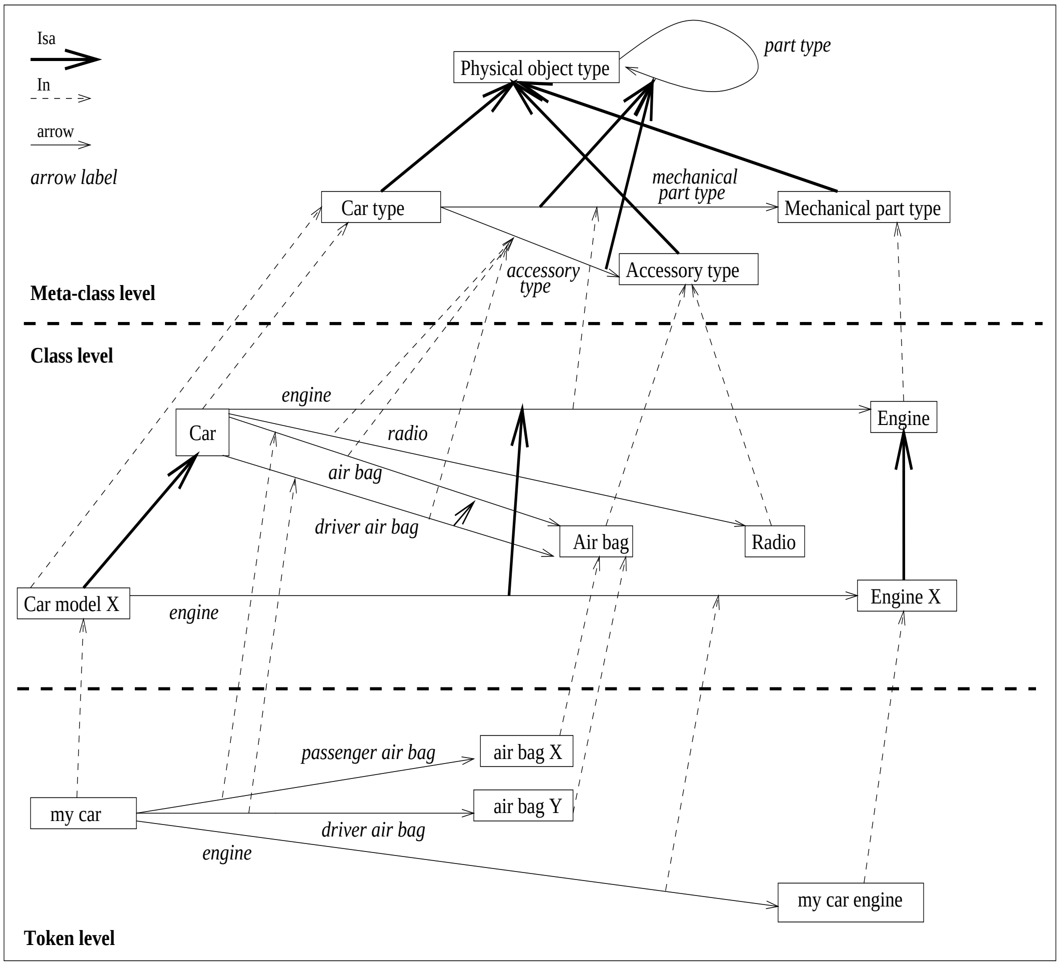

Figure 6: Part of an example information base

arrows, one from *Car* to *Engine* and the other from *Car model X* to *Engine X*. When there is no ambiguity, we identify an arrow just by using its label. Otherwise, together with the label, we use the name of the *from* object of the arrow, e.g., we say *engine* of *Car* and *engine* of *Car model X*.

At the token level, the arrow *passenger air bag* expresses the information that my car is equipped on the passenger side with the air bag X. The arrow *driver air bag* expresses the information that my car is equipped on the driver side with the air bag Y. At the class level, the arrow *air bag* expresses the information that a car may be equipped with airbags. At the meta-class[3] level, the arrow *accessory type* expresses the information that a car type may have several types of accessories.

[3]Meta-classes are classes whose instances are classes.

The token *my car* is an instance of the class *Car model X* which, in turn, is an instance of the meta-class *Car type*. The arrow token *air bag* of *my car* is an instance of the arrow class *air bag* of Car which, in turn, is an instance of the arrow meta-class *accessory type*.

The class *Engine X* is a subclass of *Engine*, expressing the fact that every engine of type X is an engine. This implies that every instance of *Engine X* is an instance of *Engine*. Similarly, the class *Car model X* is a subclass of *Car*. The arrows *mechanical part type* and *accessory type* are subclasses of the arrow *part type*, expressing the fact that mechanical part types and accessory types are part types.

The arrow *driver air bag* of Car is subclass of the arrow *air bag* of Car. Therefore, the arrow *driver air bag* of *my car*, which is an instance of *driver air bag* of Car, is also an instance of *air bag* of Car. Note that the arrows *driver air bag* and *air bag* of Car have the same $from$ and *to* objects, although they are different arrows.

4 Arrow Specialization by Restriction

In this section, we define a stronger form of the *isa* relation which we call *restriction isa* and denote *Risa*. We would like to stress that restriction isa is defined for arrow classes, only. Following our general approach, we first define restriction isa for real arrow classes and then transpose it on their representations in the model.

4.1 Restriction Isa in the Real World

We first give the definition of the restriction isa relation.

Definition 4.1 RISA **relation**
Let A, A' be two real arrow classes. We say that A is a *restriction subclass* of A', denoted by RISA(A, A'), if the following holds: (i) ISA(A, A'), and (ii) if X is an instance of A', and $from(X)$ is an instance of $from(A)$, then X is an instance of A.

Roughly speaking, if a real arrow A is a restriction subclass of a real arrow A' then the extension of A is the restriction of the extension of A' to the extension of $from(A)$.

For example, refer to Figure 7, where the real class ONLY PAINTING COLLECTOR refers to art collectors that collect only paintings. The real arrow COLLECTS of ONLY PAINTING COLLECTOR is a restriction subclass of the real arrow COLLECTS of ART COLLECTOR. This is because if an art

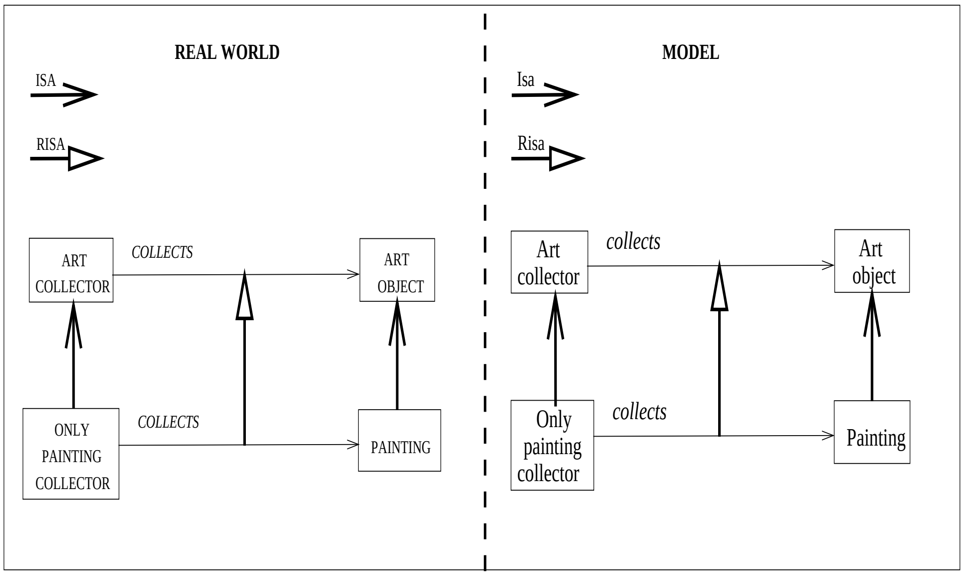

Figure 7: Example of restriction isa in the real world and in the model

collector collects an art object and the art collector happens to be an only painting collector, then the art object must be a painting. Specifically, if X is an instance of the real arrow COLLECTS of ART COLLECTOR such that $from(X)$ is an instance of ONLY PAINTING COLLECTOR then X is also an instance of the real arrow COLLECTS of ONLY PAINTING COLLECTOR.

Consider two real arrows A_1 and A_2 such that $from(A_1)$ is a subclass of $from(A_2)$ and $to(A_1)$ is a subclass of $to(A_2)$. The following proposition says that if A_1 and A_2 are restriction subclasses of the same real arrow A' then A_1 is a restriction subclass of A_2.

Proposition 4.1 Let A' be a real arrow class from a real class C' to a real class D'. Let A_1 and A_2 be any real arrow classes. If $\text{RISA}(A_1, A')$, $\text{RISA}(A_2, A')$, $\text{ISA}(from(A_1), from(A_2))$, and $\text{ISA}(to(A_1), to(A_2))$ then $\text{RISA}(A_1, A_2)$.

It follows directly from Proposition 4.1 and the ISA Fact 4 (Antisymmetry) that if the $from$ and to objects of A_1 and A_2 coincide then the real arrows A_1 and A_2 coincide. This is expressed by the following proposition.

Proposition 4.2 Let A' be a real arrow class from a real class C' to a real class D'. Let C be any subclass of C' and let D be any real class. There is at most one arrow class A from C to D such that $\text{RISA}(A, A')$.

For example, in Figure 7, the real arrow COLLECTS of ONLY PAINTING COLLECTOR is the unique real arrow from ONLY PAINTING COLLECTOR to PAINTING which is a restriction subclass of the arrow COLLECTS of ART COLLECTOR.

Note that if there is an instance X of real class A' such that $from(X)$ is an instance of real class C but $to(X)$ is not an instance of D then there is no real arrow A from C to D such that $\mathrm{RISA}(A, A')$.

4.2 Restriction Isa in the Model

Switching to the model, any two arrow classes a, a' can be related through a relation denoted by $Risa$. The pairs of arrow classes that are related through this relation are either declared by the user or derived by the system. The declaration of pairs by the user is done under the following assumption: If $Risa(a, a')$ is declared by the user then $\mathrm{RISA}(A, A')$ holds, for any arrow classes a, a' representing the real arrow classes A, A', respectively.

If $Risa(c, c')$ holds then we say that c is a *restriction subclass* of c', or that c is a *specialization by restriction* of c', or that c' is a *restriction superclass* of c.

For example, in Figure 7 (a), the arrow $collects$ of *Only painting collector* is a restriction subclass of the arrow *collects* of *Art collector*. This expresses the information that, if an art collector collects an art object and the art collector happens to be an only painting collector, then the art object must be a painting. In other words, any art object collected by an only painting collector must be a painting.

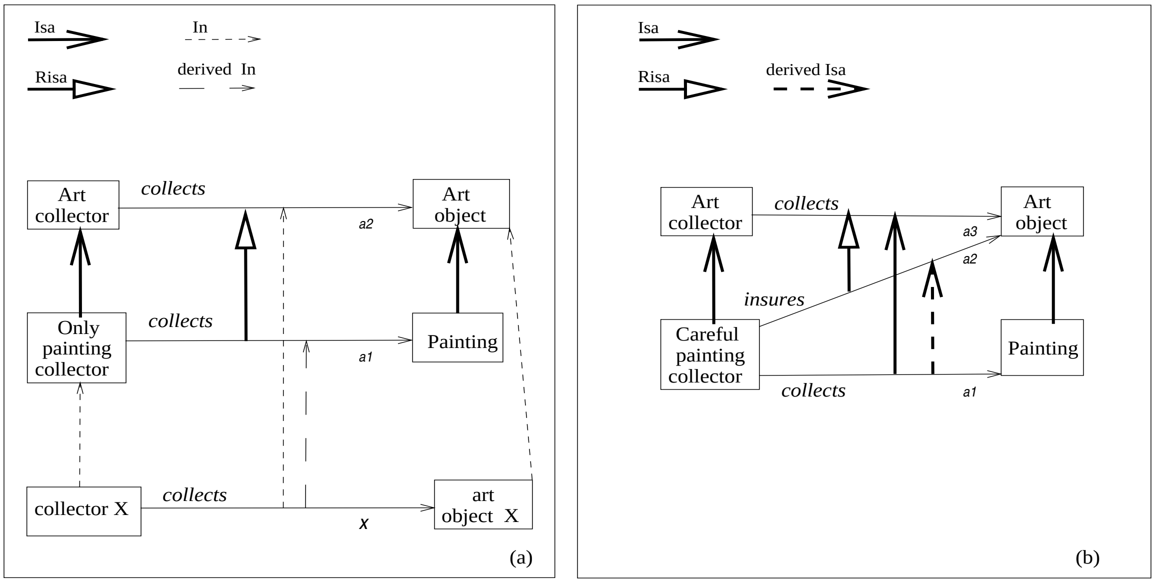

Figure 8: (a) Example for Risa Rule 2, (b) Example for Risa Rule 3

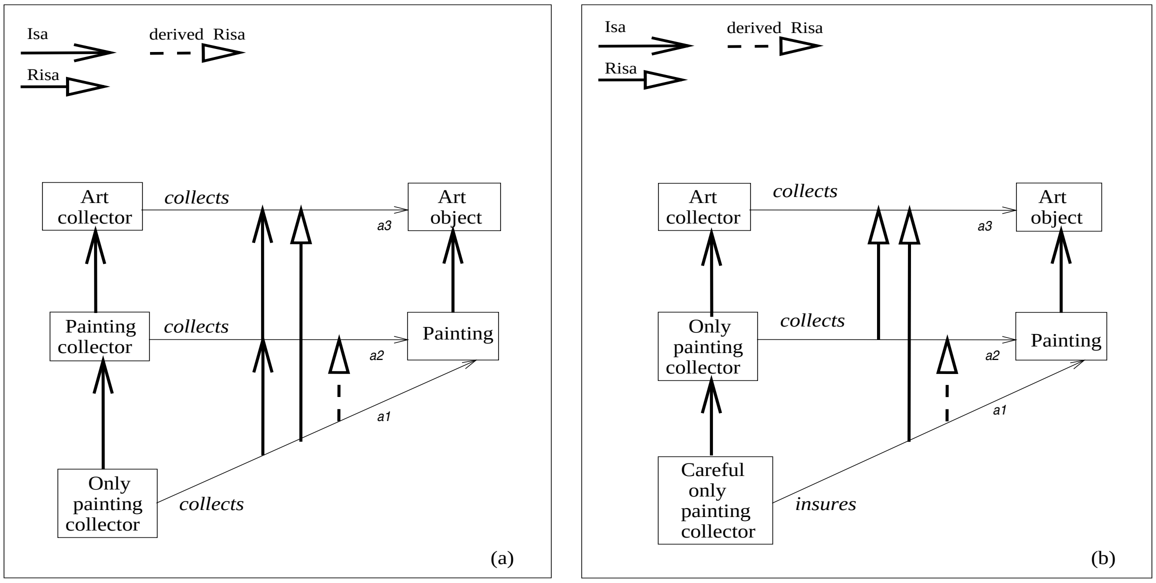

Figure 9: (a) Example for Risa Rule 5, (b) Example for Proposition 4.3

There are several rules for deriving new relations between arrows, based on given Isa and $Risa$ relations. It is important to note that these rules allow derivations between arrows at the schema level.

The soundness of the $Risa$ rules is given in the Appendix.

Risa RULES

Let x, a_1, a_2, a_3 be arrows. Then we have:

Rule 1: If $Risa(a_1, a_2)$ then $Isa(a_1, a_2)$.

Rule 2: If $Risa(a_1, a_2)$, $In(x, a_2)$, and $In(from(x),\ from(a_1))$ then $In(x, a_1)$.

Example : Referring to Figure 8 (a), suppose that the following declarations have been made: (i) the arrow *collects* of *Only painting collector* is a restriction subclass of the arrow *collects* of *Art collector*, (ii) *collector X* is an instance of *Only painting collector*, and (iii) the arrow *collects* of *collector X* is an instance of the arrow *collects* of *Art collector*. Then, it follows from *Risa* Rule 2 that the arrow *collects* of *collector X* is an instance of the arrow *collects* of *Only painting collector*.

Note that if the *art object X* is not an instance of *Painting* then the arrow instantiation constraint is violated.

Rule 3: If $Isa(a_1, a_3)$, $Risa(a_2, a_3)$, $Isa(from(a_1), from(a_2))$, and $Isa(to(a_1), to(a_2))$ then $Isa(a_1, a_2)$.

Example : The usefulness of this rule is demonstrated in Figure 8 (b). Suppose that the following declarations have been made: (i) the arrow *collects* of *Careful painting collector* is a subclass of the arrow *collects* of *Art collector*, and (ii) the arrow *insures* of *Careful painting collector* is a restriction subclass of the arrow *collects* of *Art collector*. Then, it follows from *Risa* Rule 3, that the arrow *collects* of *Careful painting collector* is a subclass of the arrow *insures* of *Careful painting collector*. This derived *Isa* relation expresses the information that all paintings collected by a careful painting collector are insured.

Rule 4: If $Risa(a_1, a_2)$, $Risa(a_2, a_3)$ then $Risa(a_1, a_3)$.

Rule 5: If $Isa(a_1, a_2)$, $Isa(a_2, a_3)$, and $Risa(a_1, a_3)$ then $Risa(a_1, a_2)$.

Example : This rule is exemplified in Figure 9 (a).

The following proposition is the counterpart of Proposition 4.1 in the model. Specifically, it shows how we can derive a *Risa* relation between two arrows a_1 and a_2 based on their *Risa* relations with a third arrow a_3.

Proposition 4.3 Let a_1, a_2, a_3 be any arrows. If $Risa(a_1, a_3)$, $Risa(a_2, a_3)$, $Isa(from(a_1), from(a_2))$, and $Isa(to(a_1), to(a_2))$ then $Risa(a_1, a_2)$.

The usefulness of this result is demonstrated in Figure 9 (b). Suppose that the following declarations have been made: (i) the arrow *insures* of *Careful only painting collector* is a restriction subclass of the arrow *collects* of *Art collector*, and (ii) the arrow *collects* of *Only painting collector* is a restriction subclass of the arrow *collects* of *Art collector*. Then it follows from Proposition 4.3, that the arrow *insures* of *Careful only painting collector* is a restriction subclass of *collects* of *Only painting collector*. This expresses the information that (i) every painting insured by a careful only painting collector has been collected, and (ii) every painting collected by a careful only painting collector is insured.

It follows directly from Proposition 4.3 and the *Isa* antisymmetry constraint that if two arrows a_1 and a_2 are restriction subclasses of the same arrow a' and the *from* and *to* objects of a_1 and a_2 coincide then the arrows a_1 and a_2 coincide. This is expressed by the following proposition which is the counterpart of Proposition 4.2.

Proposition 4.4 Let a' be an arrow class from a class c' to a class d'. Let c be any subclass of c' and let d be any class. Then, there is at most one arrow class a from c to d such that $Risa(a, a')$.

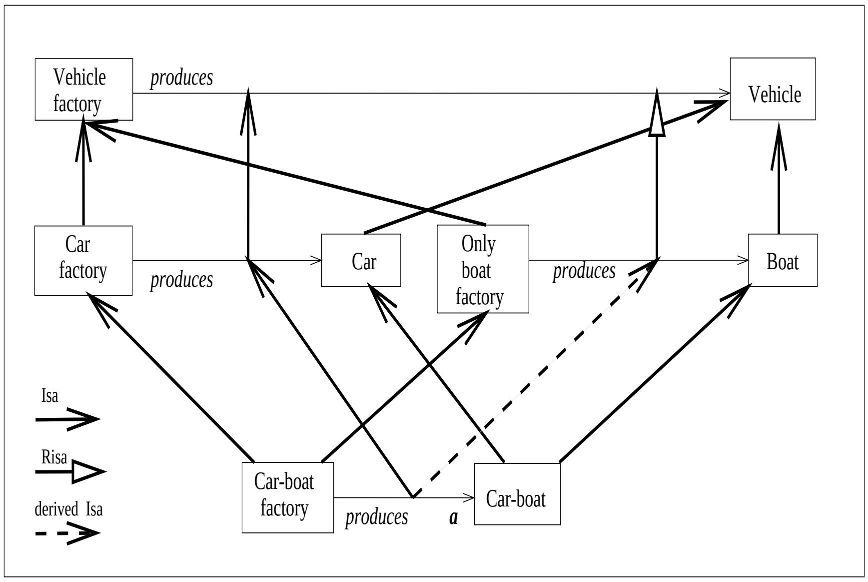

Figure 10: Example of Isa relation derivation

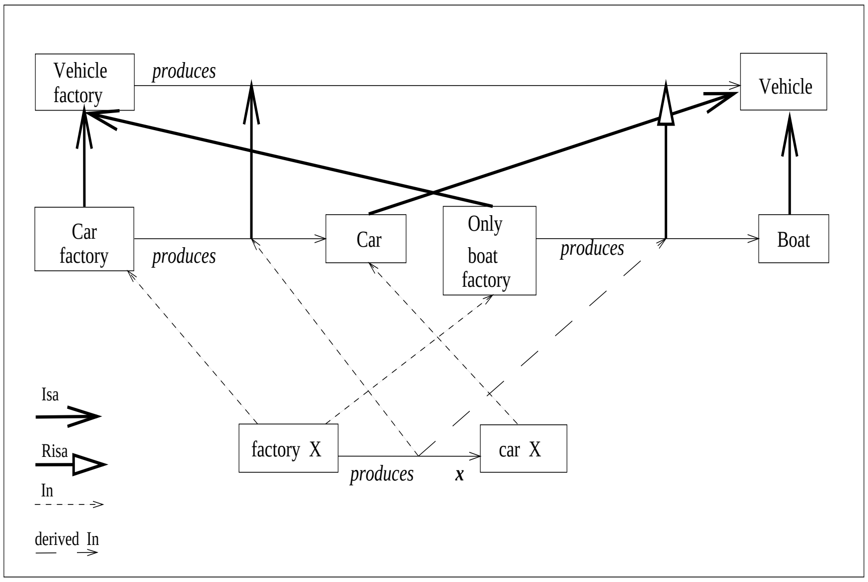

Figure 11: Example of In relation derivation

So far we have seen examples where an Isa relation is derived based on Isa and $Risa$ relations between three different arrows. Let us now see an example where a new Isa

relation is derived based on *Isa* and *Risa* relations between four different arrows. The new *Isa* relation is derived by applying both *Isa* and *Risa* rules. In Figure 10, suppose that the following declarations have been made: (i) the arrow *produces* of *Car factory* is a subclass of the arrow *produces* of *Vehicle factory*, (ii) the arrow *produces* of *Only boat factory* is a restriction subclass of the arrow *produces* of *Vehicle factory* (expressing that an only boat factory produces only boats), (iii) *Car-boat factory* is a subclass of both *Car factory* and *Only boat factory*, and (iv) the arrow *produces* of *Car-boat factory* (denoted by a) is a subclass of the arrow *produces* of *Car factory*. From the *Isa* Rule 3 (transitivity), it follows that the arrow a is subclass of the arrow *produces* of *Vehicle factory*. Then, from *Risa* Rule 3, the arrow a is also subclass of the arrow *produces* of *Only boat factory*.

Note that starting from the fact that arrow a is subclass of the arrow *produces* of *Car factory*, we derived that a is subclass of the arrow *produces* of *Only boat factory*. This is an interesting *Isa* relation derivation, as the arrows *produces* of *Car factory* and *produces* of *Only boat factory* are not related with the *Isa* relation.

Let us now see an example where an *In* relation is derived based on *Isa* and *Risa* relations between four different arrows. The *In* relation is derived by applying both *Isa* and *Risa* rules. We refer to Figure 11 where the classes *Vehicle factory*, *Car factory*, and *Only boat factory* and their arrows are as in Figure 10. Suppose that the following additional declarations have been made: (i) object *factory X* is instance of both *Car factory* and *Only boat factory*, and (ii) arrow *produces* of *factory X* is an instance of arrow *produces* of *Car factory*. From *Isa* Rule 1, it follows that arrow x is instance of arrow *produces* of *Vehicle factory*. From *Risa* Rule 2, it follows that arrow x is also instance of arrow *produces* of *Only boat factory*.

Note that starting from the fact that arrow x is instance of the arrow *produces* of *Car factory*, we derived that x is instance of the arrow *produces* of *Only boat factory*. This is an interesting *In* relation derivation as the arrows *produces* of *Car factory* and *Only boat factory* are not related with the *Isa* relation.

Note that the *to* object of the arrow *produces* of *factory X* is *car X*. As the arrow *produces* of *factory X* is instance of the arrow *produces* of *Only boat factory*, object *car X* should be instance of *Boat*. If *car X* is not an instance of *Boat* then the arrow instantiation constraint is violated. In this case a warning is given to the user indicating that the object *factory X* is possibly wrongly declared as instance of *Only boat factory*, or arrow *produces* of *Only boat factory* is possibly wrongly declared as restriction subclass of arrow $produces$ of *Vehicle factory*, or both. If none of these holds then *car X* should be declared as an instance of $Boat$.

As we have already indicated in subsection 3.4, constraints that hold in the real world can be translated in the model either into inference rules or into constraints. Therefore, instead of the arrow instantiation constraint, the following inference rule could have been specified: if an arrow a_1 is an instance of an arrow a_2 then infer that $from(a_1)$ and $to(a_1)$ are instances of $from(a_2)$ and $to(a_2)$, respectively. In this example, in particular, an In relation between *car X* and *Boat* could have been derived based on the In relation between the arrows $produces$ of *factory X* and $produces$ of *Only boat factory*. Yet, if such In relation derivations between the $from$ and to objects of arrows are made, not only erroneous declarations will remain unnoticed but also the effect of these erroneous declarations will be propagated into the information base.

5 Arrow Inheritance

In this section, we first define arrow inheritance in the real world, using the RISA relation, and then transpose the inheritance concept to the model. As properties are assigned to entities through arrows, arrow inheritance expresses property inheritance.

5.1 Arrow Inheritance in the Real World

We view arrow inheritance from a class to a subclass as arrow restriction to the subclass. In other words, the inherited arrow is an arrow of the subclass that is related to the original arrow through the RISA relation. Before we give formal definitions, we present our rationale.

Let A' be a real arrow class from a real class C' to a real class D'. Let C be a subclass of C'. Our purpose is to define the real arrow inherited by C from A'.

Recall that the real arrow A' expresses a property from C' to D' that refers collectively to a set, say E', of real arrows. The $from$ objects of arrows in E' are instances of C' and their to objects are instances of D'. For example, referring to Figure 12, let A' be the real arrow COLLECTS, C' be ART COLLECTOR, D' be ART OBJECT, and C_1 be ONLY PAINTING COLLECTOR. Here, A' expresses a property of art collectors. Thus, A' refers to a set E' of real arrows going $from$ instances of ART COLLECTOR to instances of ART OBJECT. As we can see in the figure,

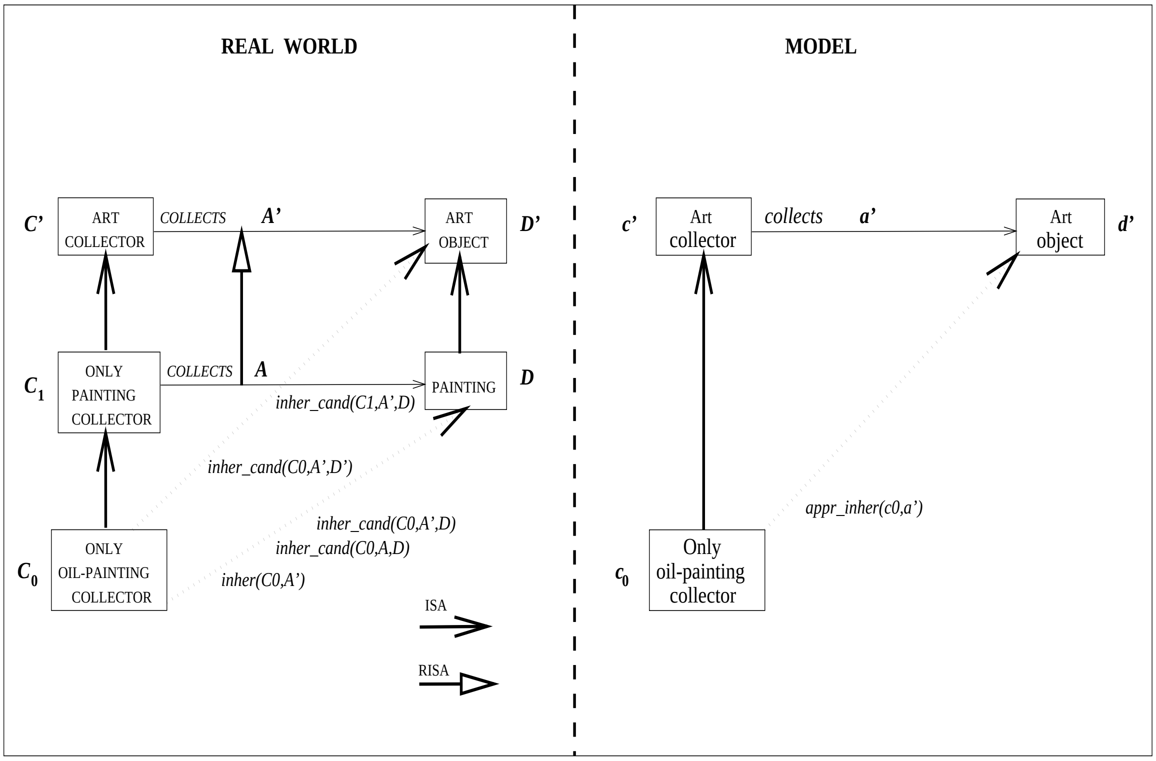

Figure 12: Inheritance candidates in the real world and in the model

C_1 is a subclass of C' and refers to only painting collectors.

The question is what part of A' expresses a property of C_1. It is this part that we shall call the arrow inherited by C_1 from A'. Obviously, this part is the set E of real arrows in E' whose $from$ object is an instance of C_1. Thus, the arrow inherited by C_1 from A', should refer collectively to the arrows in E. In other words, the arrow inherited by C_1 from A' should have as extension the set E. Continuing with our example, the part of COLLECTS that expresses a property of ONLY PAINTING COLLECTOR is the set of real arrows in the extension of COLLECTS whose $from$ object is an instance of ONLY PAINTING COLLECTOR.

Having determined the extension E of the inherited arrow, it remains to determine the *to* object of the inherited arrow. Obviously, there are many real classes D that can be this *to* object, one of them being D' itself. Intuitively, any real arrow from C_1 to a class D, referring collectively to the real arrows in E, represents information inherited by C_1 from A'. We refer to these real arrows as *inheritance candidate arrows*. In our example, ART OBJECT could be the *to* object of the real arrow inherited by ONLY PAINTING COLLECTOR from the real arrow COLLECTS of ART COLLECTOR. However, there may also be subclasses of ART OBJECT that can play this role, for example, the class PAINTING.

Among the inheritance candidate arrows, we consider as the inherited arrow the one whose *to* object gives the most specific information. Let D_{min} be the real object that refers collectively to the *to* objects of the real arrows in E. Obviously, D_{min} gives the most specific information and must be the *to* object of the inherited arrow.

Following this reasoning, we define the inheritance candidate arrows as follows.

Definition 5.1 Inheritance Candidate Arrow

Let A' be a real arrow class from a real class C' to a real class D' and let C be a subclass of C'. Let D be any real class that satisfies the following condition:

> For every instance X of A' such that $from(X)$ is an instance of C, $to(X)$ is an instance of D.

We define $inher\_cand(C, A', D)$ to be the real arrow class from C to D whose extension consists of all instances of A' whose $from$ object is an instance of C. If D is a class that does not satisfy the above condition, then $inher\_cand(C, A', D)$ is undefined. If $inher\_cand(C, A', D)$ is defined then we call it an *inheritance candidate arrow.*

It follows from the arrow equality assumption that if $inher\_cand(C, A', D)$ is defined then it is unique. Thus, $inher\_cand$ can be seen as a partial function that associates a given triple (C, A', D) with at most one real arrow.

Note that if D is a superclass of D' then $inher\_cand(C, A', D)$ is always defined. In particular, $inher\_cand(C, A', D')$ is defined.

For example, referring to Figure 12, let A' be the real arrow COLLECTS of ART COLLECTOR and D' be the real class ART OBJECT. Let A be the real arrow COLLECTS of ONLY PAINTING COLLECTOR, C_1 be the real class ONLY PAINTING COLLECTOR, and D be the real class PAINTING. Additionally, let C_0 be the class ONLY OIL-PAINTING COLLECTOR. Then, the real arrow $inher\_cand(C_0, A', D')$ is defined. This follows directly from the fact that D' is the *to* object of A'.

Consider now any instance X of A' whose $from$ object is an instance of ONLY PAINTING COLLECTOR. As real arrow A is a restriction subclass of A', the real arrow X is also an instance of A, and thus the *to* object of X is also an instance of D. Therefore, $inher\_cand(C_1, A', D)$ is defined. Note that A and $inher\_cand(C_1, A', D)$ have the same $from$ and *to* objects, and the same extension. Therefore, (from the arrow equality assumption) they coincide. It can be easily seen that $inher\_cand(C_0, A', D)$ is also defined.

Definition 5.2 Candidate Class Set
Let A' be a real arrow class from a real class C' to a real class D'. Let C be a subclass of C'. We define $cand\_class(C, A')$ to be the set of real classes D such that the real arrow $inher\_cand(C, A', D)$ is defined. We refer to these real classes as *candidate classes* and to the set $cand\_class(C, A')$ as the *candidate class set* for C and A'.

As we have already mentioned, among the inheritance candidate arrows, we would like to consider as the inherited arrow the one whose *to* object gives the most specific information.

Definition 5.3 Inherited Arrow
Let A' be a real arrow class from a real class C' to a real class D'. Let C be a subclass of C'. Let D_{min} be the least class (w.r.t. ISA) of $cand\_class(C, A')$. We define the *arrow inherited* by C from A' to be the real arrow $inher\_cand(C, A', D_{min})$ and we denote it by $inher(C, A')$.

Consider the class whose extension is the set of *to* objects of all instances X of A' such that $from(X)$ is an instance of C. It is easy to see that this class is the least class of $cand\_class(C, A')$. Thus, the least class of $cand\_class(C, A')$ always exists and $inher(C, A')$ is well defined.

We would like to give the following propositions concerning inherited arrows.

Proposition 5.1 Let A' be a real arrow class from a real class C' to a real class D' and let C be a subclass of C'. Let D be any subclass of D' such that $inher\_cand(C, A', D)$ is defined. Then, $inher\_cand(C, A', D)$ is a restriction subclass of A'.

A direct consequence of Proposition 5.1 is the following proposition which says that the inherited arrow is a restriction subclass of the original arrow.

Proposition 5.2 Let A' be a real arrow class from a real class C' to a real class D' and let C' be a subclass of C'. Then, $inher(C, A')$ is a restriction subclass of A'.

For example, in Figure 12, the arrow inherited by ONLY OIL-PAINTING COLLECTOR from the real arrow COLLECTS of ART COLLECTOR is a restriction subclass of the real arrow COLLECTS of ART COLLECTOR.

Consider two real arrows A, A' such that $Risa(A, A')$ and class C is a subclass of $from(A)$. The following proposition says that the arrows inherited by a real class C from real arrows A, A' coincide.

Proposition 5.3 Let A' be a real arrow class from a real class C' to a real class D'. Let C be a subclass of C' and let A be any real arrow. If $Risa(A, A')$ and $Isa(C, from(A))$ then $inher(C, A)$ coincides with $inher(C, A')$.

For example, in Figure 12, the arrows inherited by ONLY OIL-PAINTING COLLECTOR from the real arrows COLLECTS of ART COLLECTOR and COLLECTS of ONLY PAINTING COLLECTOR, coincide. That is, the real arrows $inher\_cand(C_0, A', D)$ and $inher\_cand(C_0, A, D)$ coincide.

As the real arrow $inher(C, A)$ is a restriction subclass of A, the following proposition is direct consequence of Proposition 5.3.

Proposition 5.4 Let A' be a real arrow class from a real class C' to a real class D' and let C be a subclass of C'. Let A be any real arrow class. If $Risa(A, A')$ and $Isa(C, from(A))$ then $inher(C, A')$ is a restriction subclass of A.

For example, in Figure 12, the arrow inherited by ONLY OIL-PAINTING COLLECTOR from the real arrow COLLECTS of ART COLLECTOR is a restriction subclass of the real arrow COLLECTS of ONLY PAINTING COLLECTOR. That is, $inher(C_0, A')$ is a restriction subclass of A.

The following theorem gives an important property of arrow inheritance. Specifically, it says that if the arrow inherited by a real class C from a real arrow A' is a restriction subclass of a real arrow A then the arrow inherited by C from A' coincides with the arrow inherited by C from A.

Theorem 5.1 Let A' be a real arrow class from C' to D' and let C be a subclass of C'. Let A be any real arrow class. If $Risa(inher(C, A'), A)$ then $inher(C, A)$ coincides with $inher(C, A')$.

5.2 Arrow Inheritance in the Model

Switching to the model, let a' be an arrow class from a class c' to a class d' and let c be a subclass of c'. Let A', C be the real objects represented by a', c, respectively. We have seen that the real arrow $inher(C, A')$ is actually the inheritance candidate arrow $inher\_cand(C, A', D_{min})$, where D_{min} is the least class in the set $cand\_class(C, A')$. Let $inher(c, a')$ be the arrow that represents the real arrow $inher(C, A')$. Obviously, the *from* object of $inher(c, a')$ is c and the *to* object of $inher(c, a')$ is the class representing D_{min}.

However, as we have explained, only some of the real classes in $cand\_class(C, A')$ and only some of their ISA relations are represented in the model. This implies that D_{min} may not be represented in the model. Thus, the question is how to choose an arrow that "approximates" $inher(c, a')$. To this end, we proceed roughly as follows:
First we compute a set of classes $cand\_class(c, a')$, called the *candidate class set* for c and a'. Intuitively, a class d is in $cand\_class(c, a')$ if we "know" that D is in $cand\_class(C, A')$. Then, we define the arrow approximating $inher(c, a')$ based on the set $cand\_class(c, a')$.

We begin by giving a set of inference rules that allow us to compute the set $cand\_class(c, a')$. The soundness of these rules is proved in the Appendix.

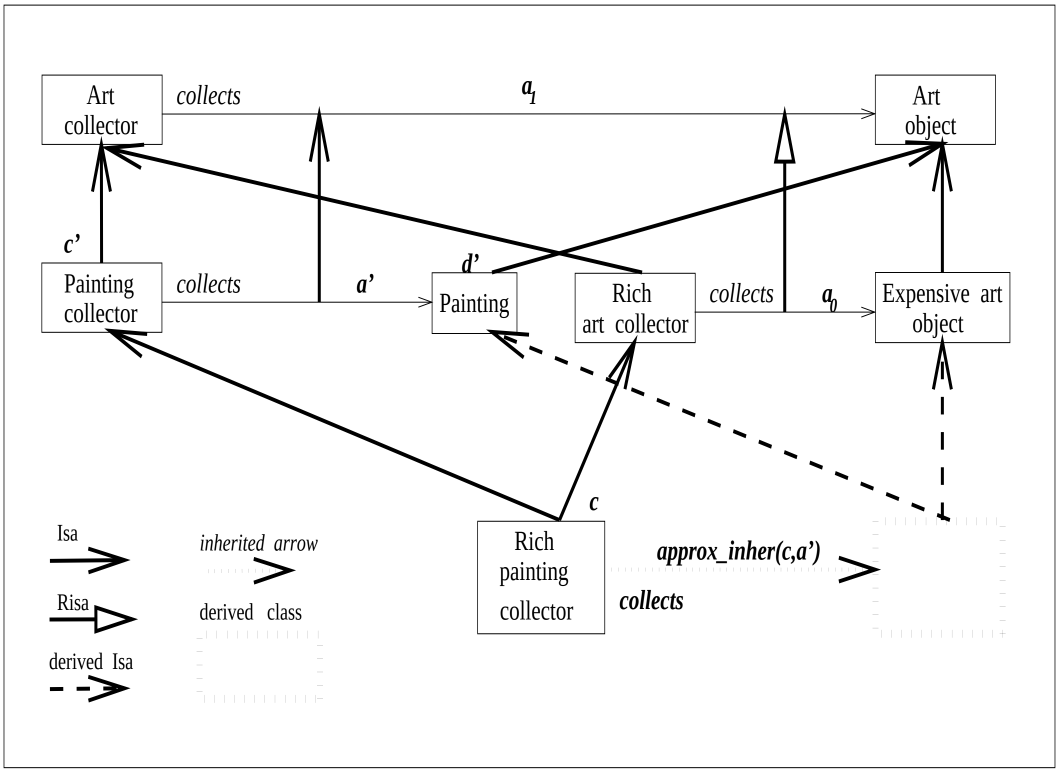

Figure 13: Example for the inheritance rules

INHERITANCE RULES

Let a' be an arrow from class c' to class d' and let c be a subclass of c'. Let a, a_0, a_1 be any classes.

Rule 1 $Risa(inher(c, a'),\ a')$.

This rule says that any arrow inherited from an arrow class a' is a restriction subclass of a'.

Rule 2 If $Isa(inher(c, a'), a_1)$, $Risa(a_0, a_1)$, and $Isa(c, from(a_0))$ then $Isa(inher(c, a'),\ a_0)$.

This rule says that if (i) the arrow inherited by a class c from an arrow a' is a subclass of an arrow a_1, (ii) an arrow a_0 is a restriction subclass of a_1, and (iii) class c is subclass of $from(a_0)$, then the arrow inherited by c from a' is subclass of arrow a_0.

Example: This rule is exemplified in Figure 13. First, observe that arrow $inher(c, a')$ is a restriction subclass of a' (Inheritance Rule 1). Additionally, observe that a' is subclass of a_1. Thus, from Isa Rule 3 (transitivity), we derive that $inher(c, a')$ is subclass of a_1. On the other hand, arrow a_0 is restriction subclass of a_1. As c is a subclass of $from(a_0)$, we have all three conditions of Inheritance Rule 2 satisfied. Thus, we obtain that $inher(c, a')$ is subclass of a_0.

Rule 3: If $Isa(inher(c, a'),\ a)$ then (i) $Isa(to(inher(c, a')),\ to(a))$, and (ii) $to(a)$ is in $cand\_class(c, a')$.

This rule says that if an inherited arrow is subclass of an arrow class a then its *to* object is subclass of $to(a)$, and thus $to(a)$ is in $cand\_class(c, a')$.

Example: Continuing with our previous example (Figure 13), as $inher(c, a')$ is a subclass of a', the *to* object of $inher(c, a')$ is a subclass of d'. Thus, d' is in $cand\_class(c, a')$. As $inher(c, a')$ is a subclass of a_1, the *to* object of $inher(c, a')$ is a subclass of $to(a_1)$ and thus, $to(a_1)$ is in $cand\_class(c, a')$. Additionally, as $inher(c, a')$ is a subclass of a_0, $to(a_0)$ is in $cand\_class(c, a')$. Thus, $cand\_class(c, a')$ consists of the objects d', $to(a_0)$, and $to(a_1)$.

Rule 4: If $Isa(inher(c, a'), inher(c, a))$ and $Isa(inher(c, a), inher(c, a'))$ then $inher(c, a')$ coincides with $inher(c, a)$.

Using the Inheritance Rules just presented, we can compute the set of classes $cand\_class(c, a')$. We then use the set $cand\_class(c, a')$ in order to define the arrow that "approximates" $inher(c, a')$. We denote the approximating arrow by $appr\_inher(c, a')$.

Roughly speaking, the basic idea is to take the least class of $cand\_class(c, a')$ as the *to* object of $appr\_inher(c, a')$. However, as not all classes of $cand\_class(C, A')$ are represented in the model, the set $cand\_class(c, a')$ may *not* have a least class. Therefore, in our definition of $appr\_inher(c, a')$, we distinguish two cases:

1. If there is a class which is the least class of $cand\_class(c, a')$ then we take that class as the *to* object of $appr\_inher(c, a')$.

2. Otherwise, we create in the model a new class as well as new Isa relations such that the new class *becomes* the greatest lower bound of $cand\_class(c, a')$.

Definition 5.4 Inherited arrow

Let a' be an arrow class from a class c' to a class d'. Let c be a subclass of c'. Let A', C be the real objects represented by the objects a', c, respectively. The arrow $appr\_inher(c, a')$, called the *arrow inherited* by c from a', is defined in three steps as follows:

Step 1 Compute $cand\_class(c, a')$ using the Inheritance Rules.

Step 2 Let S be the set of real classes represented by the classes in $cand\_class(c, a')$ and let H_{min} be the greatest lower bound (with respect to ISA) of S. We define a class h_{min} as follows:

Case 1: If the set $cand\_class(c, a')$ has a least class (w.r.t. Isa) then h_{min} is this class.

Case 2: Otherwise, a *new* class h_{min} is created in the model which represents H_{min} and which is related to other classes through Isa relations as follows:

(i) h_{min} is subclass of each class in $cand\_class(c, a')$,

(ii) if a class d is subclass of each class in $cand\_class(c, a')$ then d is also subclass of h_{min}.

Step 3 Define $appr\_inher(c, a')$ to be the arrow from c to h_{min} that represents the real arrow $inher(C, A', H_{min})$.

We would like to make a few remarks concerning the definition of inherited arrow.

In Step 1, d' is obviously in $cand\_class(c, a')$. So, the set $cand\_class(c, a')$ is never empty. Moreover, let x be a class in $cand\_class(c, a')$. Then, all superclasses of x are in $cand\_class(c, a')$, because of the Isa transitivity rule. On the other hand, some subclasses of x may be in $cand\_class(c, a')$, because of the Inheritance Rule 2 (see example of Inheritance

Rule 2). Therefore, in general, the set $cand\_class(c, a')$ will contain classes of three kinds: superclasses of d', subclasses of d', or classes not related to d' through the Isa relation. For example, referring to Figure 13, it is not difficult to see that

$$cand\_class(c, a') = \{\textit{Painting, Art object, Expensive art object}\}$$

Note that *Art object* is a superclass of d' (i.e., of $Painting$), whereas *Expensive art object* is neither superclass nor subclass of d'.

In Step 2, we distinguish two cases in the definition of h_{min}. In Case 1, h_{min} gives the best approximation to the real class D_{min} that can be provided by the model. Here, by best approximation we mean the most specific class in the model for which we "know" that represents a superclass of D_{min}. For example, referring to Figure 12, suppose that the *to* object of $inher(C0, A')$ is PAINTING. As PAINTING is not represented in the model, the set $cand\_class(c0, a')$ contains only *Art object*. Thus h_{min}, in this example, is *Art object* and this is the best approximation to PAINTING. Moreover, $appr\_inher(c0, a')$ is the arrow from $c0$ to *Art object* and represents the real arrow $inher(C0, A', D')$. We view $appr\_inher(c0, a')$ as the best approximation to the real arrow $inher(C0, A')$.

In Case 2 of Step 2, the set $cand\_class(c, a')$ does not have a least class. Therefore, we cannot give a best approximation to the class D_{min} using the classes currently available in the model. We are thus led to create a new class h_{min}. In order for h_{min} to be a good approximation of D_{min}, the real class represented by h_{min} should satisfy the following requirements:

1. it should be subclass of each class in S (this is because D_{min} is subclass of each class in S),
2. it should be superclass of D_{min} (in order for the definition of inherited arrow in the model to be sound).

As D_{min} may coincide with the greatest lower bound of S, the greatest lower bound of S is the only class that always satisfies these requirements. This is why in our definition we take h_{min} to represent the greatest lower bound of S.

The position of h_{min} in the Isa hierarchy (determined by (i) and (ii) in the definition), reflects the position of H_{min} in the ISA hierarchy. Note that after executing (i) and (ii), h_{min} is the greatest lower bound of the set $cand\_class(c, a')$.

For example, referring to Figure 13, we have seen earlier that

$$cand\_class(c, a') = \{Painting,\ Art\ object,\ Expensive\ art\ object\}$$

Note that $cand\_class(c, a')$ has no least class. So we are led to introduce a new class h_{min} and to relate it to the existing classes so that h_{min} becomes subclass of both *Painting* and *Expensive art object.*

We would like to emphasize an important property of inherited arrows: the *to* object of an inherited arrow in the model always represents a superclass of the *to* object of the inherited arrow in the real world.

A general comment concerning Definition 5.4 is the following: In Case 2 of Step 2, it is possible that the set $cand\_class(c, a')$ has no least class but has greatest lower bound, call it g. Then, one may wonder why not define h_{min} to be the class g. The answer is that the real class represented by g may not be the greatest lower bound of S, and in fact, it may not even be related though ISA to D_{min} (see also the discussion concerning Figure 19 in Section 6).

As we will see below (theorems 5.3 and 5.4), the $Risa$ and Isa relations of $inher(c, a')$ are transferred to $appr\_inher(c, a')$. The following proposition is a special case of Proposition 4.3, for deriving a $Risa$ relation between $inher(c, a')$ and an arrow a_2. Note that unlike Proposition 4.3, no condition on the *to* objects of $inher(c, a')$ and a_2 is required.

Proposition 5.5 Let a' be an arrow from class c' to class d' and let c be a subclass of c'. Let a_2, a_3 be arrow classes. If $Risa(inher(c, a'),\ a_3)$, $Risa(a_2, a_3)$, and $Isa(c, from(a_2))$ then $Risa(inher(c, a'),\ a_2)$.

We claim that our choice of approximating $inher(C, A')$ by $appr\_inher(c, a')$ is a reasonable one. We base this claim on the fact that $appr\_inher(c, a')$ satisfies several of the properties satisfied by $inher(C, A')$, as we shall show now. The following proposition is the counterpart of Proposition 5.2.

Proposition 5.6 Let a' be an arrow from c' to d' and let c be a subclass of c'. Let A' be the real arrow represented by a. Then, the arrow $appr\_inher(c, a')$ represents a real object which is a restriction subclass of A'.

The following proposition is the counterpart of Proposition 5.3.

Proposition 5.7 Let a' be an arrow from c' to d' and let c be a subclass of c'. Let a be any arrow class. If $Risa(a, a')$ and $Isa(c, from(a))$ then $appr\_inher(c, a)$ coincides with $appr\_inher(c, a')$.

To illustrate Proposition 5.7, refer to Figure 13. As arrow a_0 is a restriction subclass of a_1, and class c is a subclass of $from(a_0)$, the inherited arrows $appr\_inher(c, a_0)$ and $appr\_inher(c, a_1)$ coincide.

The following theorem is the counterpart of Theorem 5.1.

Theorem 5.2 Let a' be an arrow from c' to d' and let c be a subclass of c'. Let a be any arrow. If $Risa(inher(c, a'),\ a)$ then the inherited arrows $appr\_inher(c, a)$ and $appr\_inher(c, a')$ coincide.

Let A be the real arrow represented by a. As the real arrow represented by $appr\_inher(c, a)$ is a restriction subclass of A (Proposition 5.6), Theorem 5.3 below is a direct consequence of Theorem 5.2, and expresses an important property: if $inher(c, a')$ is restriction subclass of an arrow a then $appr\_inher(c, a')$ is also restriction subclass of a.

Theorem 5.3 Let a' be an arrow from c' to d' and let c be a subclass of c'. Let a be any arrow and let A be the real arrow represented by a. If $Risa(inher(c, a'),\ a)$ then the real arrow represented by $appr\_inher(c, a')$ is a restriction subclass of A.

The following theorem expresses the important property that if $inher(c, a')$ is a subclass of an arrow a then $appr\_inher(c, a')$ is also a subclass of a.

Theorem 5.4 Let a' be an arrow from c' to d' and let c be a subclass of c'. Let a be any arrow and let A be the real arrow represented by a. If $Isa(inher(c, a'),\ a)$ then the real arrow represented by $appr\_inher(c, a')$ is a subclass of A.

Let a' be an arrow class from class c' to class d', with label l. Let c be a subclass of c'. A problem arises as to the label to be used for the arrow inherited by c from a'. This problem is not treated here. However, for the purposes of this paper, we can consider l as a label of the inherited arrow, used only for display purposes[4].

Note that the arrows inherited by a class c from two or more different arrows may coincide. Thus, the dislay label of an inherited arrow may not be uniquely defined. For

[4]Display labels of inherited arrows are indicated in the figures in bold, italic letters

the purposes of this paper, we have chosen to give to the inherited arrow all the labels of the arrows from which it is inherited. For example in Figure 15, the arrows inherited by *Careful only painting collector* from the arrows *insures* of *Art collector* and *collects* of *Only painting collector* coincide, and the resulting arrow receives two display labels, i.e., *collects* and *insures*.

5.3 Examples of Inheritance

We will now present more complex examples that demonstrate the usefulness of our approach to arrow inheritance.

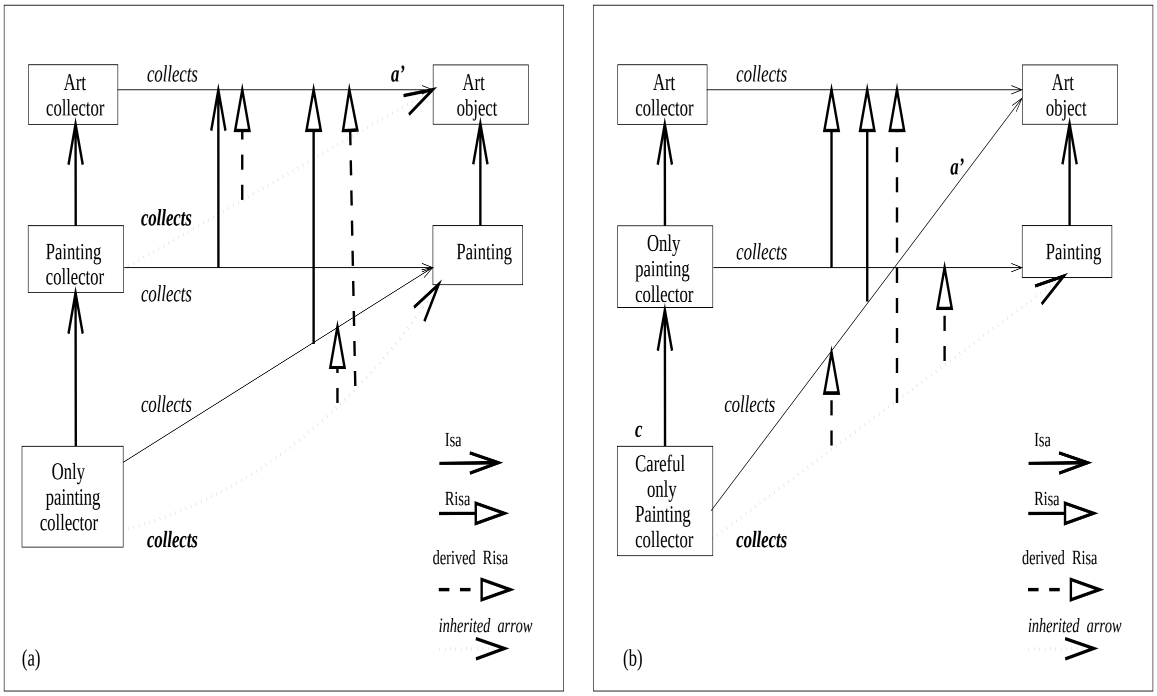

Figure 14: Examples of inherited arrows that refine the *to* object of the original arrow

For the first example, refer to Figure 14 (a). We consider two cases of inheritance from the arrow *collects* of *Art collector* (denoted by a'): (i) the inheritance of a' by *Painting collector*, and (ii) the inheritance of a' by *Only painting collector*.

In the first case, the *to* object of the arrow inherited by *Painting collector* from a' is the same as that of the original arrow a'. This is because the candidate-class set for *Art collector* and a' is $\{Art\ object\}$. Note that *Art object* is put into the candidate-class set after successive applications of Inheritance Rule 1 and Inheritance Rule 3. In this case, our

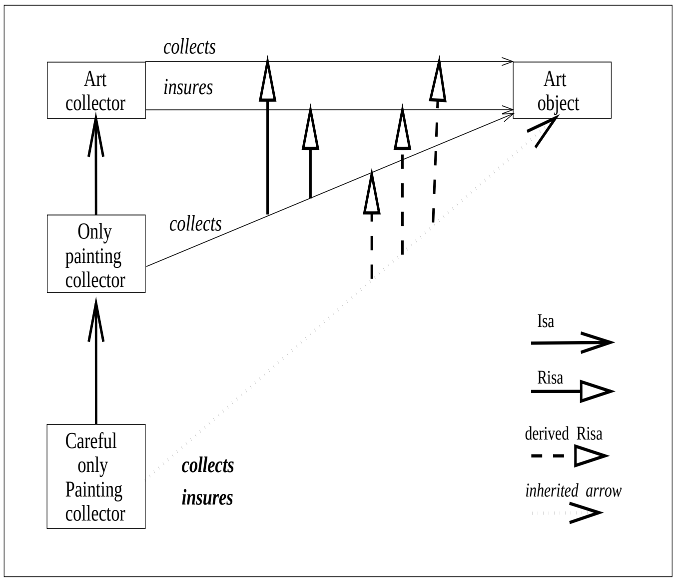

Figure 15: Example of inherited arrows that coincide

inheritance mechanism provides the following information about the class *Painting collector*: painting collectors may collect art objects other than paintings. Note that the display label of the inherited arrow is *collects*, i.e., the label of the arrow from which it was inherited.

In the second case, the *to* object of the arrow inherited by *Only painting collector* from a' is *Painting* which is a subclass of the *to* object of a'. This is because the candidate class set for *Only painting collector* and a' is $\{$*Art object*, *Painting*$\}$. Note that *Painting* is put into the candidate-class set after successive applications of Inheritance Rule 1, Inheritance Rule 2, and Inheritance Rule 3. In this case, our inheritance mechanism provides the following information about the class *Only painting collector*: only-painting collectors do not collect art objects other than paintings.

Note that our inheritance mechanism behaves differently in the two cases. The difference is due to the fact that the arrow *collects* of *Painting collector* has been declared as subclass of a', whereas the arrow *collects* of *Only painting collector* has been declared as a restriction subclass of a'. In other words, through the *Risa* relation the user can differentiate between possible semantics for particular classes, as it was claimed in the Introduction.

As another example, referring to Figure 14 (b), the class *Careful only painting collector* (denoted by c) has an explicitly declared arrow *collects* (denoted by a') whose *to* object is *Art object*. Based on the explicitly declared *Risa* relations, it follows from Proposition 5.5 and Theorem 5.3 that the arrow inherited by c from *collects* of *Art collector* is a restriction

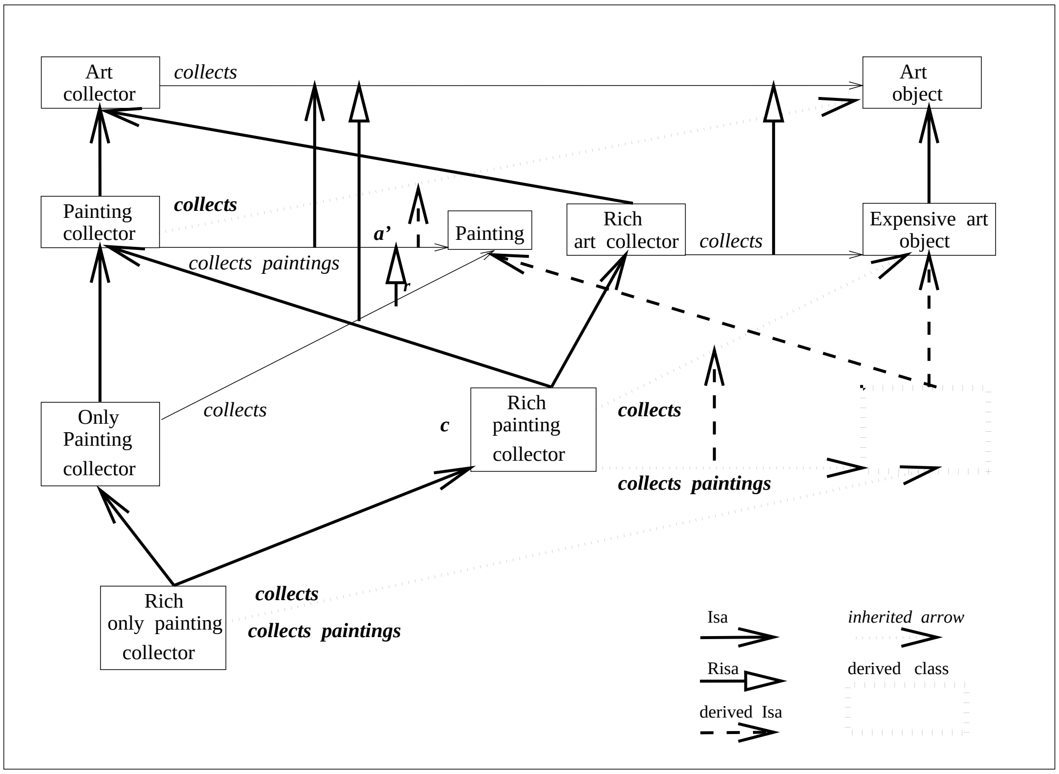

Figure 16: Example of inherited arrows with derived *to* object

subclass of a'. Additionally, it follows from Proposition 5.7 that the arrow inherited by c from *collects* of *Art collector* coincides with the arrow inherited by c from a'. This inherited arrow gives the information that careful only painting collectors do not collect art objects other than paintings. We see in this example that the arrow inherited by a class c from an arrow a' can give a finer *to* value than that of a' even in the case that a' is an arrow of c.

Referring to Figure 15, the class *Careful only painting collector* does not have any explicitly declared arrow. Yet, it inherits from the arrows *collects* and *insures* of *Art collector* and from the arrow *collects* of *Only painting collector*. We would like to point out that all three inherited arrows coincide due to the explicitly declared *Risa* relations (as shown in the figure). Note that the single inherited arrow has two display labels.

In Figure 16 [5], arrow *collects* of *Rich art collector* is a restriction subclass of *Art collector*, expressing the information that rich art collectors collect only expensive art objects. *Rich painting collector* (denoted by c) is a subclass of both *Painting collector* and *Rich art collector*. The arrows inherited by c from the arrows *collects* of *Art collector* and *collects*

[5] We do not show that *Painting* is a subclass of *Art Object* in order not to further complicate the figure.

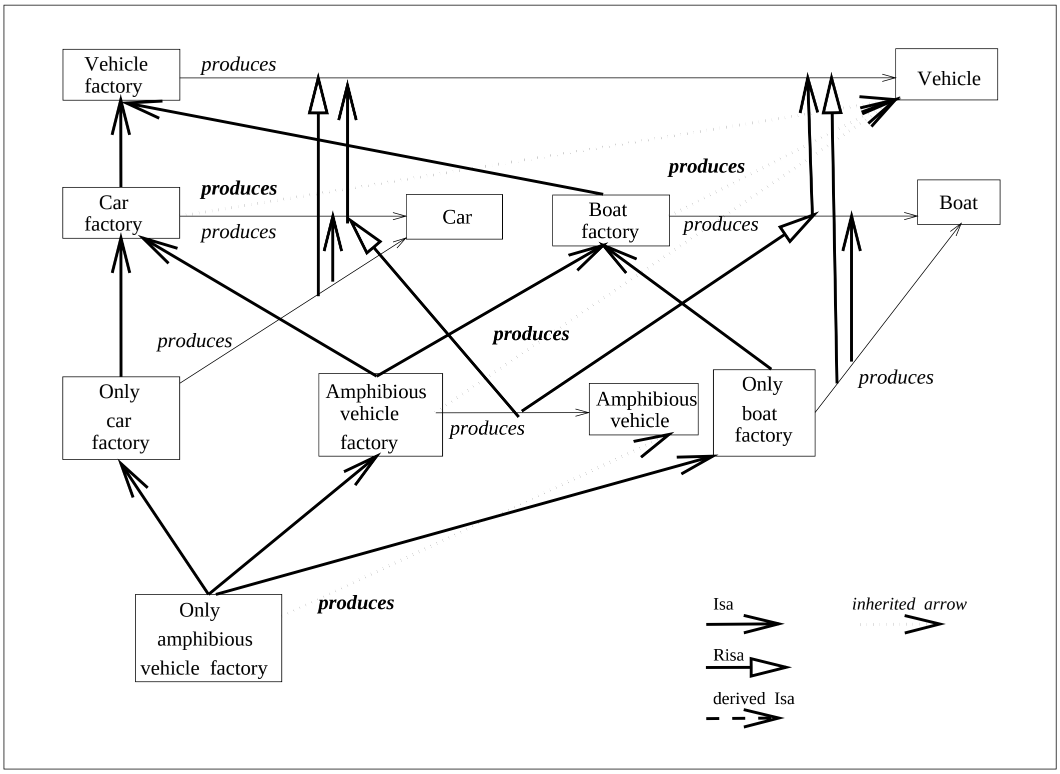

Figure 17: Examples of inherited arrows

of *Rich art collector* coincide and constitute a single derived arrow.

The arrow inherited by c from *collects paintings* of *Painting collector* (denoted by a') points to a derived object which is subclass of *Painting* and *Expensive art object*. This is because $inher(c, a')$ is a restriction subclass of a', and thus subclass of *collects* of *Art collector*. From Inheritance Rule 2, it is also subclass of arrow *collects* of *Rich art collector*. Thus, the candidate class set for c and a' is $\{$*Painting*, *Art object*, *Expensive art object*$\}$. Because this set has no least class, the *to* object of the inherited arrow is a new class which is the greatest lower bound of classes *Painting* and *Expensive art object*. Note that from Inheritance Rule 2, an *Isa* relation is derived between the arrows inherited by c having display labels *collects paintings* and *collects*.

In the same example, *Rich only painting collector* is subclass of both *Only painting collector* and *Rich painting collector*. All arrows inherited by *Rich only painting collector* coincide and point to the new class.

In Figure 17 [6], we see that a car factory produces cars and possibly other vehicles.

[6] We do not show that *Car* and *Boat* are subclasses of *Vehicle* and that *Amphibious vehicle* is a subclass

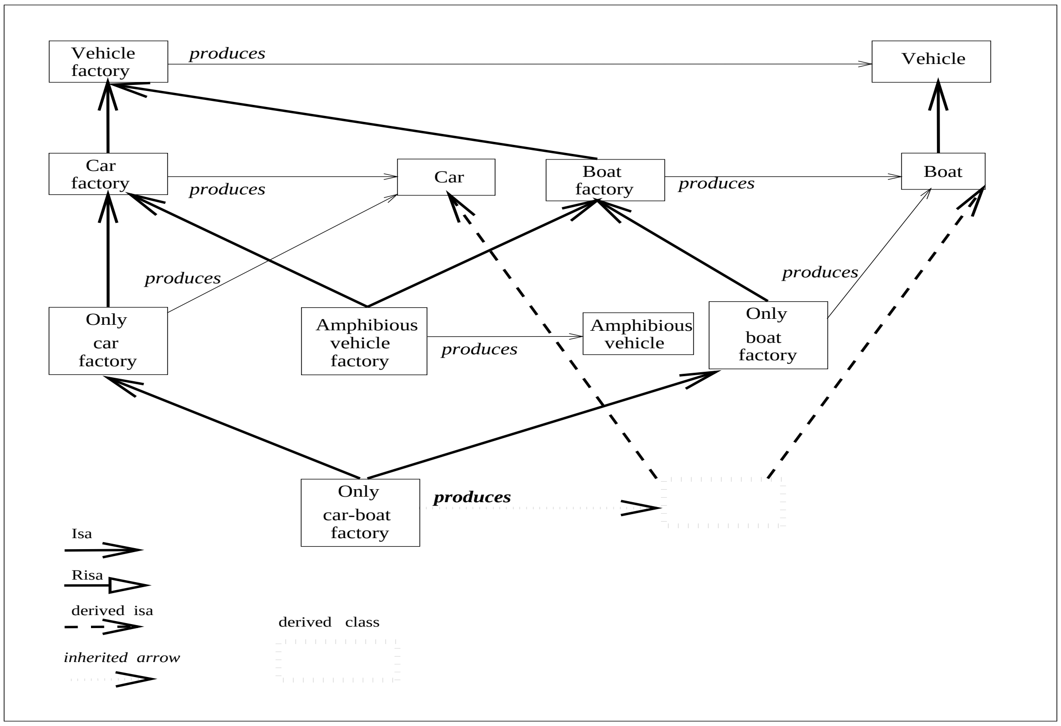

Figure 18: Example of inherited arrows with derived *to* object

Similarly, a boat factory produces boats and possibly other vehicles. The arrow *produces* of *Amphibious vehicle factory* is a restriction subclass of both arrows *produces* of *Car factory* and *produces* of *Boat factory*, expressing that all cars and all boats that an amphibious vehicle factory produces, are amphibious. As car factories and boat factories may produce vehicles other than cars and boats, the arrow inherited by *Amphibious vehicle factory* from the arrow *produces* of *Car factory* points to $Vehicle$. This indicates that an amphibious vehicle factory possibly produces vehicles that are not amphibious vehicles.

Class *Only amphibious vehicle factory* is subclass of *Only car factory*, *Only boat factory*, and *Amphibious vehicle factory*. Because an only car factory produces only cars and an only boat factory produces only boats, an only amphibious vehicle factory produces only amphibious vehicles. Note that all arrows inherited by *Only amphibious vehicle factory* coincide and point to *Amphibious vehicle*.

In Figure 18, *Only car-boat factory* is subclass of *Only car factory* and *Only boat factory*, but not of *Amphibious vehicle factory*. Similarly to *Only amphibious vehicle factory* of Figure 17, all arrows inherited by *Only car-boat factory* coincide to a single inherited arrow.

---

of both Car and $Boat$.

However, in contrast with the example of Figure 17, this inherited arrow points to a derived class which is subclass of both Car and $Boat$.

6 Comparison with Related Work

It is quite common to assign a set of properties to objects via the *type* concept, which is also intimately connected with the semantics of property inheritance. To establish a framework for the comparison of our model with related ones, we briefly review the type concept and we distinguish two categories of data models with respect to their treatment of property inheritance.

In object-oriented systems and some extensible systems [21, 9], a type is a set of properties and types are related through the subtype relationship in a type hierarchy [7]. A type T is a *subtype* of a type T' iff T supports all properties of T' with the same or more refined value domain (property refinement). T may have additional properties. Properties of T that appear in supertypes of T are called *inherited*, whereas properties of T that do not appear in any of the supertypes of T are called *local*.

Classes correspond to collections of objects and are related through the subclass relationship in a class hierarchy. Each class C has a set of properties (local or inherited) and a unique most specific type, called *type* of C. The properties of a class are exactly those of its type. If a class C is a subclass of C' then the type of C should be a subtype of the type of C'.

In our model, the concept of property refinement corresponds to the concept of property specialization by restriction ($Risa$). Let T be a subtype of T'. If a property p of T is related to a property p' of T' through the $Risa$ relation then we can say that p refines the property inherited from p'. By contrast, if p is not related with any of the properties of T' through the $Risa$ relation then p is considered a property additional to those of T'.

The type of a class in a class hierarchy depends on the semantics of property inheritance. We distinguish data models with respect to property inheritance into two kinds: *extension* data models and *refinement* data models. In extension data models, all property definitions within a class are considered to define local properties of the class and not to refine inherited properties. Thus, the type of a class C is the union of the local properties of C and the local properties of all superclasses of C. In refinement data models, property definitions within a class either refine inherited properties (by restricting their value domain) or define local properties of the class.

Let C' be a class with a property p' and C be a subclass of C'. In extension data models, the only way that p can be related to p' through the *Isa* or *Risa* relation is by expressing their relation using a constraint language (outside the inheritance mechanism). Examples of extension data models are the extended ER model [28], the extended UI model [26][7] and the "Living in a Lattice" data model [16].

In refinement data models, on the other hand, properties inherited by a subclass may be refined by restricting the value domain of the original property. If a property p of a class C refines a property p' of a superclass C' of C then the relationship between p and p' is expressed in our model by the *Risa* relation. However, in refinement data models, the only way that p can be related to p' through the *Isa* relation is by expressing this relation using a constraint language (outside the inheritance mechanism). For example (see Figure 1), the class *Art collector* has a property *collects* and its subclass *Painting collector* also has a property *collects*. The semantics that refinement data models give to *Painting collector* is that painting collectors collect only paintings. As we have explained in the Introduction, this is not the intended semantics for *Painting collector*.

In refinement data models, the type of a class can be determined by visiting the classes in the hierarchy as follows: (i) First, visit all classes that have no superclasses, (ii) after visiting all classes of the previous step, continue with their immediate subclasses with the constraint that a class is visited only when all of its superclasses have already been visited.

Assume that we currently visit a class C. Obviously, the type of the immediate superclasses of C is already known. The class C inherits all properties with distinct origin[8] of its immediate superclasses.

If two such inherited properties have the same name then the inherited properties should be renamed. As in this paper we do not deal with property name conflicts, we treat renamed inherited properties similarly to other inherited properties. If two properties p' and p'' of an immediate superclass of C have the same origin then refinement data models usually adopt

[7] In this work, the authors propose strategies for resolving property name ambiguities appearing in queries.

[8] If a property p is inherited from a property p' and p' is inherited from a property p'' then we say that p is transitively inherited from p''. If a property p is transitively inherited from a local property p'' then we say that the *origin* of p is p''.

one of the following approaches:

Approach 1: Only one of p' and p'' is inherited. The selection of the inherited property is done automatically or is left to the user. Such data models are the ORION data model [3] and the O_2 data model [14].

Approach 2: One or both of p' and p'' are inherited. Such data model is the ODE data model [2].

Approach 3: The constraint is enforced that p' and p'' must have identical value domains. If this constraint is satisfied then the properties inherited by class C from p' and p'' coincide and the value domain of the inherited property is the common value domain of p' and p''. Such data model is the POSTGRES data model [25, 27].

Approach 4: The properties inherited by class C from p' and p'' coincide and the value domain of the inherited property is the greatest lower bound of the value domains of p' and p'' (in the existing class hierarchy). These data models force the class hierarchy to be a lattice, so that the greatest lower bound of the value domains of p' and p'' always exists. This approach is followed by the deductive object-oriented systems LOGIN [1] and COMPLEX [8].

Approach 5: The properties inherited by class C from p' and p'' coincide and the value domain of the inherited property is not a unique class but a set consisting of the value domains of p' and p''. Legal values of the inherited attribute should be instances of each class in its value domain. This approach is followed by traditional frame-based models [15, 4] and by the deductive object-oriented language F-logic [20].

The value domain of an inherited property refined in class C is as specified in the refinement declaration. Having determined the properties inherited by C and their value domains, the type of C is the set of properties inherited by C union the local properties of C.

The problem with Approach 1 is that as only one of p' and p'' is inherited by class C, the type of C may not be a subtype of the type of one of its superclasses. For example, in Figure 19, the class *Only car-boat factory* has two immediate superclasses: *Only car factory* and *Only boat factory*. Additionally, the property *produces* of *Vehicle factory* is the common origin of the properties *produces* of *Only car factory* (denoted by p') and *produces* of *Only boat factory* (denoted by p''). If *Only car-boat factory* inherits only property p' then the

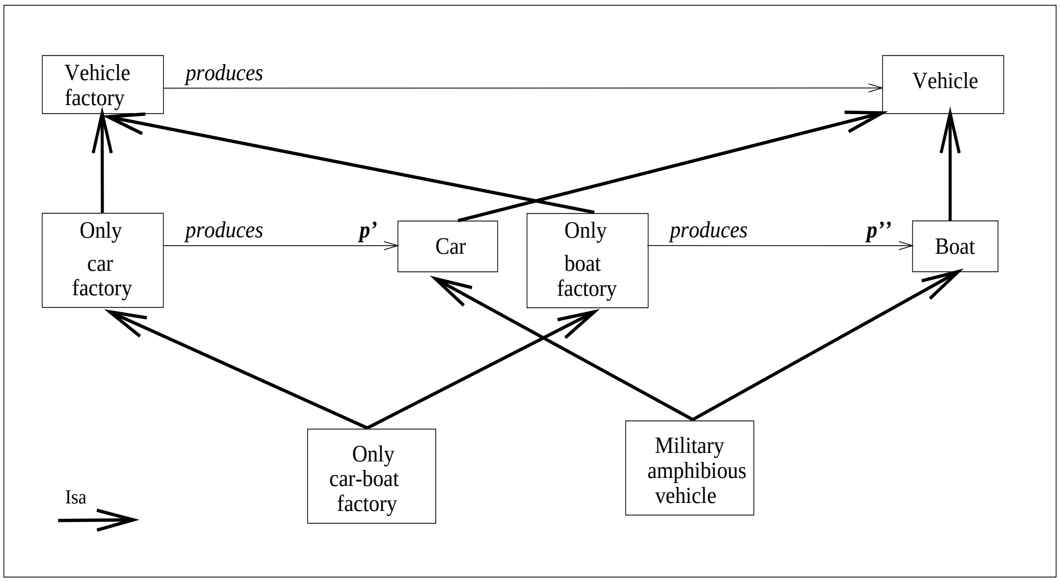

Figure 19: Example of property inheritance when properties have the same origin

type of *Only car-boat factory* is not a subtype of *Only boat factory*. Thus, it is possible to have instances of *Only car-boat factory* that produce cars which are not boats.

Approach 2 presents the same problem with Approach 1, if only one of p' and p'' is inherited by class C. If both of p' and p'' are inherited by C then the corresponding inherited properties are considered to be independent though they semantically correspond to the same property. For example, in Figure 19, the class *Only car-boat factory* will have one property inherited from p' and a different property inherited from p''. This implies that an only car-boat factory can produce cars that are not boats and boats that are not cars. Clearly, this is not the intended semantics for *Only car-boat factory*.

Though Approach 3 does not have the subtyping problems of approaches 1 and 2, the enforced constraint for p' and p'' (identical value domains) is too strict. For example, in Figure 19, the only way that *Only car-boat factory* is allowed to be subclass of both *Only car factory* and *Only boat factory* is by defining a property *produces* of *Only car-boat factory* whose value domain is a subclass of both Car and $Boat$. In the case that no class exists that corresponds to this value domain, one should be created by the user just for this reason.

In Approach 4, the value domain of the inherited property is the greatest lower bound of the value domains of p' and p'' in the existing class hierarchy. However, this greatest lower bound class may not be a superclass of the value domain of the inherited property in the real world. Thus, in contrast to our model, the value domain of the inherited property may not be a superclass of the value domain of the inherited property in the real world. To illustrate Approach 4, refer to Figure 19. As the greatest lower bound of Car and *Boat* is *Military amphibious vehicle*, the value domain of the property inherited by *Only car-boat factory* from the properties p' and p'' will be *Military amphibious vehicle*. However, the intended semantics for *Only car-boat factory* is to produce amphibious vehicles (possibly not military). In our model, the value domain of the inherited property would have been a new class which is subclass of both Car and *Boat* and a strict superclass of *Military amphibious vehicle*.

In Approach 5, the value domain of the inherited property is a set of classes. This approach is closer to our model than the other approaches. However, in our model, the value domain of an inherited property is always a single class with a position in the class hierarchy. To illustrate Approach 5, refer to Figure 19. The value domain of the property inherited by *Only car-boat factory* from the properties p' and p'' is the set $\{Car, Boat\}$. This implies that valid values for the inherited property should be instances of both Car and *Boat*. Note that, as the value domain of the inherited property is a set of classes, the inherited property cannot be treated similarly to local properties.

The KL-ONE system [4] supports the *RoleSet Differentiation* construct, which is close to the Isa relation in our model. Additionally, KL-ONE supports the *RoleSet Restriction* construct, which is close to the $Risa$ relation of our model. With regard to *RoleSet Restriction*, KL-ONE follows Approach 5. Yet, in contrast to Isa and $Risa$ relations in our model, *RoleSet Differentiation* and *RoleSet Restriction* do not interact to relate the inherited properties with other properties and to refine the value domain of the inherited properties.

Figure 20 illustrates an example that compares our approach to property inheritance with other models. In our model (see Figure 20 (a)), the property *collects paintings* of *Painting collector* is a subclass of the property *collects* of *Art collector*. The property *collects* of *Only oil-painting collector* is a restriction subclass of property *collects* of *Art collector*. From our inference rules, we derive that the value domain of the property inherited by *Only oil-painting collector* from the property *collects paintings* of *Painting collector* is *Oil painting*. By contrast, in other models (see Figure 20 (b)), the value domain of the inherited property is *Painting*.

For another example, refer to Figure 2. In our model, the value domain of the property

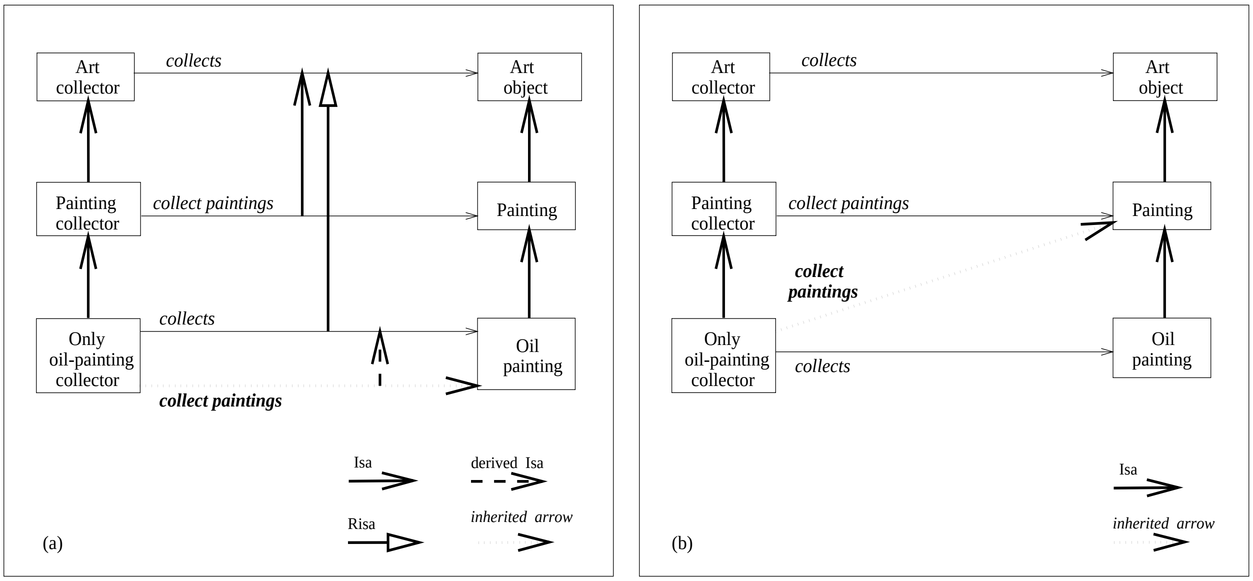

Figure 20: (a) Property inheritance in our model, (b) Property inheritance in other models

inherited by *Rich painting collector* is a new class which is the greatest lower bound of *Painting* and *Expensive art object*. In contrast, in other models, the value domain of the inherited property is *Painting*.

Finally, we would like to emphasize that several rich data models recently proposed, e.g. PSM [17] and ORM [5], do not provide formalizations of property inheritance.

7 Conclusions

In this paper, we have studied a particular form of knowledge inference, namely, property inheritance. Our motivation comes from the fact that when the properties of a class are semantically related, confusion may arise as to what the class really means.

Conventional systems cope with the problem by imposing a priori one of possibly many interpretations for the class. Our approach is to support multiple semantics and let the user choose the intended semantics for each class. To this end, we have proposed constructs that provide support for different semantics of inheritance. The inference rules supporting these constructs provide automatic refinement of the value domain of the inherited properties and useful information at schema level. Our approach is based on a clear separation between the real world, where information is considered to be complete, and the model where information is incomplete. In fact, the formal justification of the inference rules is based on relations that hold in the real world.

In this paper, properties are inherited from classes to subclasses. However, property inheritance can also take place from a class to its instances. This kind of inheritance is called *instance inheritance*. For example, assume that class *Art collector* has a property *collects* with value domain *Art object* and let p denote this property. Every instance o of *Art collector* can instance-inherit a property from p indicating that o collects art objects. A possible value domain of the instance-inherited property is *Art object*. However, this value domain can be refined based on relations of p with other properties. Other information declared by the user, may be utilized for this purpose. We currently investigate (i) new structures allowing the user to express information about the value domains of properties of o (the exact property values may be unknown), and (ii) inference rules that will support these new structures and lead to refinement of the value domain of the instance-inherited properties.

In this paper, we have followed the classical logic view of the real world, according to which In and Isa relations can always be evaluated to 0 (false) or 1(true). However, this may not always be possible. For example, non-boolean logics, such as fuzzy logic [29, 30], take a more general approach by considering that constraint satisfaction is associated with a degree of truth between 0 (false) and 1(true). Further work is needed to extend our model to account for the non-boolean logic view of the real world.

Acknowledgement: The authors wish to thank Dr. Martin Doerr for stimulating discussions and comments on earlier versions of the paper.

APPENDIX: Proofs

In the proofs, the real object represented by an object in the model is denoted by the same symbol as that of the model object but with the letters in capital.

ISA FACTS: Proofs

Let O be any real object and C, C', C'' be any real classes.

Fact 1 If ISA(C, C') and IN(O, C) then IN(O, C').

Proof As ISA(C, C'), the extension of C is a subset of the extension of C'. Thus, from the fact that O is an instance of C, it follows that O is an instance of C'. $\diamond$

Fact 2 ISA(C, C).

Proof It follows directly from the definition of the ISA relation. $\diamond$

Fact 3 If ISA(C, C') and ISA(C', C'') then ISA(C, C'').

Proof First we consider the case where C, C', C'' are real individual classes. As ISA(C, C') and ISA(C', C'') the extension of C is a subset of the extension of C''. Thus, it holds that ISA(C, C'').

Next we consider the case where C, C', C'' are real arrow classes. From the facts ISA(C, C') and ISA(C', C''), we derive that (i) ISA$(from(C), from(C''))$, (ii) ISA$(to(C), to(C''))$, and (iii) the extension of C is a subset of the extension of C''. Thus, it holds that ISA(C, C''). $\diamond$

Fact 4 If ISA(C, C') and ISA(C', C) then C coincides with C'.

Proof First we consider the case where C, C' are real individual classes. As ISA(C, C') and ISA(C', C) the extension of C equals the extension of C'. Thus, from the individual equality assumption, C coincides with C'.

Next we consider the case that C, C' are real arrow classes. From the facts ISA(C, C') and ISA(C', C), we derive that (i) $from(C)$ coincides with $from(C')$, (ii) $to(C)$ coincides with $to(C')$, and (iii) the extension of C equals the extension of C'. Thus, from the arrow equality assumption, C coincides with C'. $\diamond$

Isa RULES: Soundness proofs

The soundness of the Isa rules follows directly from the ISA Facts.

Proposition 4.1 Let A' be a real arrow class from a real class C' to a real class D'. Let C be any subclass of C' and A_1, A_2 be any real arrow classes. If $\mathrm{RISA}(A_1, A')$, $\mathrm{RISA}(A_2, A')$, $\mathrm{ISA}(from(A_1), from(A_2))$ and $\mathrm{ISA}(to(A_1), to(A_2))$ then $\mathrm{RISA}(A_1, A_2)$.

Proof We will first show that the arrow classes A_1, A_2 have the same extension. Let X be an instance of A_1. From the fact that $\mathrm{RISA}(A_1, A')$, it follows that X is an instance of A'. As $\mathrm{ISA}(from(A_1), from(A_2))$, it follows that $from(X)$ is an instance of $from(A_2)$. From the fact that $\mathrm{RISA}(A_2, A')$, it follows that X is an instance of A_2. From the facts that $\mathrm{ISA}(from(A_1), from(A_2))$, $\mathrm{ISA}(to(A_1), to(A_2))$, and the extension of A_1 is subset of the extension of A_2, it follows that $\mathrm{ISA}(A_1, A_2)$.

Let X be an instance of A_2 whose $from$ object is an instance of $from(A_1)$. From the fact that $\mathrm{RISA}(A_2, A')$, it follows that X is an instance of A'. From the facts that $\mathrm{RISA}(A_1, A')$, X is an instance of A', and $from(X)$ is an instance of A_1, it follows that X is an instance of A_1. Thus, all conditions of Definition 4.1 (RISA) hold and Proposition 4.1 follows. $\diamond$

Risa RULES: Soundness proofs

Let x, a_1, a_2, a_3 be arrows. Then we have:

Rule 1 If $Risa(a1, a2)$ then $Isa(a1, a2)$.

Proof It follows directly from condition (i) in the Definition 4.1 ($Risa$). $\diamond$

Rule 2 If $Risa(a_1, a_2)$, $In(x, a_2)$ and $In(from(x),\ from(a_1))$ then $In(x, a_1)$.

Proof It follows directly from condition (ii) in the Definition 4.1 ($Risa$). $\diamond$

Rule 3 If $Isa(a_1, a_3)$, $Risa(a_2, a_3)$, $Isa(from(a_1), from(a_2))$, and $Isa(to(a_1), to(a_2))$ then $Isa(a_1, a_2)$.

Proof Let A_1, A_2, A_3 be the real object represented by objects a_1, a_2, a_3, respectively. If X is an instance of A_1 then $from(X)$ is an instance of $from(A_1)$ and thus, an instance of $from(A_2)$. As it holds that ISA(A_1, A_3), we have that X is an instance of A_3. Then, as RISA(A_2, A_3), we obtain that X is an instance of A_2. Thus, we have that ISA(A_1, A_2). $\diamond$

Rule 4 If $Risa(a_1, a_2)$, $Risa(a_2, a_3)$ then $Risa(a_1, a_3)$.

Proof Let A_1, A_2, A_3 be the real object represented by objects a_1, a_2, a_3, respectively. From the facts that $Risa(a_1, a_2)$ and $Risa(a_2, a_3)$, it follows that RISA(A_1, A_2) and RISA(A_2, A_3). Thus, from ISA Fact 3 (transitivity), it follows that ISA(A_1, A_3). Let X be an instance of A_3 such that $from(X)$ is an instance of $from(A_1)$. From RISA(A_2, A_3), we derive that X is also an instance of A_2. Then, from RISA(A_1, A_2), we derive that X is also an instance of A_1. Therefore, it holds that RISA(A_1, A_3), and thus $Risa(a_1, a_3)$. $\diamond$

Rule 5 If $Isa(a_1, a_2)$, $Isa(a_2, a_3)$, and $Risa(a_1, a_3)$ then $Risa(a_1, a_2)$.

Proof Let X be an instance of A_2 such that $from(X)$ be an instance of $from(A_1)$. From ISA(A_2, A_3), we derive that X is also an instance of A_3. Then, from RISA(A_1, A_3), we derive that X is also an instance of A_1. From this and the fact that ISA(A_1, A_2), it follows that RISA(A_1, A_2), and thus $Risa(a_1, a_2)$. $\diamond$

Proposition 4.3 Let a_1, a_2, a_3 be any arrows. If $Risa(a_1, a_3)$, $Risa(a_2, a_3)$, $Isa(from(a_1), from(a_2))$, and $Isa(to(a_1), to(a_2))$ then $Risa(a_1, a_2)$.

Proof From $Risa$ Rule 3, we derive that $Isa(a_1, a_2)$. Then, from $Risa$ Rule 5, we derive that $Risa(a_1, a_2)$. $\diamond$

Proposition 5.1 Let A' be a real arrow class from a real class C' to a real class D', and let C be a subclass of C'. Let D be any subclass of D' such that $inher\_cand(C, A', D)$ is defined. Then, $inher\_cand(C, A', D)$ is a restriction subclass of A'.

Proof We will first show that $inher\_cand(C, A', D)$ is a subclass of A'. From our assumptions, we have ISA(C, C') and ISA(D, D'). Additionally, the extension of $inher\_cand(C, A', D)$ is subset of the extension of A'. Thus, $inher\_cand(C, A', D)$ is a subclass of A'. The fact that $inher\_cand(C, A', D)$ is a restriction subclass of A' follows now from Definition 4.1 (RISA) and Definition 5.1 (inheritance candidate arrow).

Proposition 5.3 Let A' be a real arrow class from a real class C' to a real class D'. Let C be a subclass of C' and let A be any real arrow. If $Risa(A, A')$ and $Isa(C, from(A))$ then $inher(C, A)$ coincides with $inher(C, A')$.

Proof We will first show that the extensions of the real arrows $inher(C, A)$ and $inher(C, A')$ coincide. Let X be an instance of $inher(C, A')$. Obviously, X is an instance of A' and $from(X)$ is an instance of C. From the fact that $Risa(A, A')$, it follows that X is an instance of A and thus an instance of $inher(C, A)$. Going the other way, let X be an instance of $inher(C, A)$. Obviously, X is an instance of A and $from(X)$ is an instance of C. From the fact that $Isa(A, A')$, it follows that X is an instance of A' and thus an instance of $inher(C, A')$.

Recall that the *to* object of an inherited arrow refers collectively to the *to* objects of the real arrows in the extension of the inherited arrow. Therefore, the *to* objects of $inher(C, A)$ and $inher(C, A')$ coincide. Proposition 5.3 now follows from the arrow equality assumption. $\diamond$

Theorem 5.1 Let A' be a real arrow class from C' to D' and let C be a subclass of C'. Let A be any real arrow class. If $Risa(inher(C, A'), A)$ then $inher(C, A)$ coincides with $inher(C, A')$.

Proof We will first show that the extensions of $inher(C, A)$ and $inher(C, A')$ coincide. Let X be an instance of $inher(C, A)$. Obviously, X is an instance of A and $from(X)$ is an instance of C. From the fact that $Risa(inher(C, A'), A)$, it follows that X is an instance of $inher(C, A')$. Going the other way, let X be an instance of $inher(C, A')$. Obviously, $from(X)$ is an instance of C. As $Risa(inher(C, A'), A)$, X is instance of A. Thus, X is an instance of $inher(C, A)$.

As the *to* object of an inherited arrow refers collectively to the *to* objects of the real arrows in the extension of the inherited arrow, the *to* object of $inher(C, A)$ coincides with the *to* object of $inher(C, A')$. From the arrow equality assumption, it follows that $inher(C, A)$ coincides with $inher(C, A')$. $\diamond$

INHERITANCE RULES: Soundness proofs

Let a' be an arrow from class c' to class d' and let c be a subclass of c'. Let a, a_0, a_1 be any arrows.

Rule 1 $Risa(inher(c, a'),\ a')$.

Proof Directly from Proposition 5.2. $\diamond$

Rule 2 If $Isa(inher(c, a'),\ a_1)$, $Risa(a_0, a_1)$, and $Isa(c, from(a_0))$ then that $Isa(inher(c, a'),\ a_0)$.

Proof Let C, A', A_0, A_1 be the real objects represented by the objects c, a', a_0, a_1, respectively. Let X be an instance of $inher(C, A')$. As $Isa(inher(c, a'),\ a_1)$, it holds that $\text{ISA}(inher(C, A'),\ A_1)$. Therefore, X is an instance of A_1. As $Risa(a_0, a_1)$ and $Isa(c, from(a_0))$, it follows that $\text{RISA}(A_0, A_1)$ and $\text{ISA}(C, from(A_0))$. Thus, X is an instance of A_0 and therefore, it holds that $\text{ISA}(inher(C, A'), A_0)$. $\diamond$

Rule 3 If $Isa(inher(c, a'),\ a)$ then (i) $Isa(to(inher(c, a')),\ to(a))$ and (ii) $to(a)$ is in $cand\_class(c, a')$.

Proof Let C, A', A be the real objects represented by the objects c, a', a, respectively. It is easy to see that $inher\_cand(C, A', to(A))$ is defined. Thus, from Definition 5.2 (candidate class set), $to(A)$ is in $cand\_class(C, A')$ and $to(a)$ is inserted into $cand\_class(c, a')$. $\diamond$

Rule 4 If $Isa(inher(c, a'), inher(c, a)), Isa(inher(c, a), inher(c, a'))$ then $inher(c, a')$ coincides with $inher(c, a)$.

Proof Directly from ISA Fact 4 (antisymmetry). $\diamond$

Proposition 5.5 Let a' be an arrow from class c' to class d' and let c be a subclass of c'. Let a_2, a_3 be arrow classes. If $Risa(inher(c, a'),\ a_3)$, $Risa(a_2, a_3)$, and $Isa(c, from(a_2))$ then $Risa(inher(c, a'),\ a_2)$.

Proof From Inheritance Rule 2, it is derived that $Isa(inher(c, a), a_2)$. Then, from Inheritance Rule 3, it is derived that $Isa(to(inher(c, a)), to(a_2))$. Proposition 5.5, now follows directly from Proposition 4.3. $\diamond$

Proposition 5.6 Let a' be an arrow from c' to d' and c be a subclass of c'. Let A' be the real arrow represented by a. Then, the arrow $appr\_inher(c, a')$ represents a real object which is a restriction subclass of A'.

Proof Let h_{min} be the *to* object of $appr\_inher(c, a')$ and let H_{min} be the real object represented by h_{min}. It holds that $Isa(h_{min}, d')$ and thus, it holds that ISA(H_{min}, D'). Then, from Proposition 5.1, the real arrow $inher(C, A', H_{min})$ is a restriction subclass of A'. As the arrow $appr\_inher(c, a')$ represents the real arrow $inher(C, A', H_{min})$, Proposition 5.6 follows. $\diamond$

Proposition 5.7 Let a' be an arrow from c' to d' and let c be a subclass of c'. Let a be any arrow class. If $Risa(a, a')$ and $Isa(c, from(a))$ then $appr\_inher(c, a)$ coincides with $appr\_inher(c, a')$.

Proof To show that $appr\_inher(c, a)$ coincides with $appr\_inher(c, a')$, it is enough to show that the *to* objects of $appr\_inher(c, a)$ and $appr\_inher(c, a')$ coincide. This is because if h_{min} is this common *to* object and h_{min} represents the real object H_{min} then both $appr\_inher(c, a)$ and $appr\_inher(c, a')$ represent the real object $inher\_cand(C, A', H_{min})$.

To show that the *to* objects of $appr\_inher(c, a)$ and $appr\_inher(c, a')$ coincide, we will show that the sets $cand\_class(c, a)$ and $cand\_class(c, a')$ coincide. We will first show that $Isa(inher(c, a), inher(c, a'))$. From Inheritance Rule 1, we have that $Risa(inher(c, a), a)$. As $Risa(a, a')$, we derive from Isa Rule 3 (transitivity) that $Isa(inher(c, a), a')$. From Inheritance Rule 1, we have that $Risa(inher(c, a'), a')$. As the $from$ objects of $inher(c, a)$ and $inher(c, a')$ are the same, all conditions of Inheritance Rule 2 are satisfied. Thus, we derive that $Isa(inher(c, a), inher(c, a'))$.

We will now show that $Isa(inher(c, a'), inher(c, a))$. From Inheritance Rule 1, we have that $Risa(inher(c, a'), a')$. As $Risa(a, a')$, and $Isa(c, from(a))$, both conditions of Inheritance Rule 2 are satisfied. Thus, it is derived that $Risa(inher(c, a'), a)$. From Inheritance Rule 1, we derive that $Risa(inher(c, a), a)$. As the $from$ objects of $inher(c, a)$ and $inher(c, a')$ are the same, all conditions of Inheritance Rule 2 are satisfied. Therefore, from Inheritance Rule 2, we derive that $Isa(inher(c, a'), inher(c, a))$.

From Inheritance Rule 4, we derive that the arrows $inher(c, a)$ and $inher(c, a')$ coincide. As the only rule that defines the Candidate Class set is Inheritance Rule 3, it follows that $cand\_class(c, a)$ and $cand\_class(c, a')$ coincide.

What follows is a second proof of the fact that sets $cand\_class(c, a)$ and $cand\_class(c, a')$ coincide. This proof is longer than the first one but does not make use of Inheritance Rule 4. We have included it here as it is interesting on its own.

Let d be a class in $cand\_class(c, a)$. Then d is the *to* object of an arrow a_0 such that $Isa(inher(c, a), a_0)$. The relation $Isa(inher(c, a), a_0)$ is derived by applying a sequence $RSeq$ of the following rules: Inheritance Rule 1, Inheritance Rule 2, and Isa Rule 3 (transitivity). The first rule in $RSeq$ is Inheritance Rule 1, giving $Isa(inher(c, a), a)$.

From Inheritance Rule 1, it follows that $Risa(inher(c, a'), a')$. As $Risa(a, a')$ and $Isa(c, from(a))$, it follows from Inheritance Rule 2 that $Isa(inher(c, a'), a)$. We can easily see that, as $Isa(inher(c, a), a_0)$ is derived by the rules in $RSeq$ (starting from the relation $Isa(inher(c, a), a))$, the relation $Isa(inher(c, a'), a_0)$ is also derived. Consequently, d is in $cand\_class(c, a')$.

Let d be a class in $cand\_class(c, a')$. Then d is the *to* object of an arrow a_0 such that $Isa(inher(c, a'), a_0)$. The relation $Isa(inher(c, a'), a_0)$ is derived by applying a sequence $RSeq$ of the following rules: Inheritance Rule 1, Inheritance Rule 2, and Isa Rule 3 (transitivity). Inheritance Rule 1 is the first rule applied, giving $Isa(inher(c, a'), a')$.

From Inheritance Rule 1, it follows that $Risa(inher(c, a), a)$. As $Risa(a, a')$, it follows from Isa Rule 3 (transitivity) that $Isa(inher(c, a), a')$. We can easily see that, as $Isa(inher(c, a'), a_0)$ is derived by the rules in $RSeq$ (starting from the relation $Isa(inher(c, a'), a')$, the relation $Isa(inher(c, a), a_0)$ is also derived. Consequently, d is in $cand\_class(c, a)$. $\diamond$

Theorem 5.2 Let a' be an arrow from c' to d' and let c be a subclass of c'. Let a be any arrow. Let A be the real arrow represented by a. If $Risa(inher(c, a'),\ a)$ then the inherited arrows $appr\_inher(c, a)$ and $appr\_inher(c, a')$ coincide.

Proof To show that $appr\_inher(c, a)$ coincides with $appr\_inher(c, a')$, it is enough to show that the *to* objects of $appr\_inher(c, a)$ and $appr\_inher(c, a')$ coincide. This is because if h_{min} is this common *to* object and h_{min} represents the real object H_{min} then both $appr\_inher(c, a)$ and $appr\_inher(c, a')$ represent the real object $inher\_cand(C, A', H_{min})$.

To show that the *to* objects of $appr\_inher(c, a)$ and $appr\_inher(c, a')$ coincide, we will show that the sets $cand\_class(c, a)$ and $cand\_class(c, a')$ coincide.

We will first show that $Isa(inher(c, a), inher(c, a'))$. From Inheritance Rule 1, it is derived that $Risa(inher(c, a), a)$. Note that *from* object of $inher(c, a)$ coincides with the the *from* object of $inher(c, a')$. As it holds (from our assumptions) that $Risa(inher(c, a'), a)$, all conditions of Inheritance Rule 2 are satisfied. Thus, it follows that $Isa(inher(c,a), inher(c,a'))$ holds.

We will now show that $Isa(inher(c,a'), inher(c,a))$. It holds (from our assumptions) that $Risa(inher(c,a'),a)$. From Inheritance Rule 1, we derive that $Risa(inher(c,a),a)$. Note that $from$ object of $inher(c,a)$ coincides with the $from$ object of $inher(c,a')$. As all conditions of Inheritance Rule 2 are satisfied, it follows that $Isa(inher(c,a'), inher(c,a))$ holds.

From Inheritance Rule 4, it is derived that arrows $inher(c,a)$ and $inher(c,a')$ coincide. As the only rule that defines the Candidate Class set is Inheritance Rule 3, it follows that $cand\_class(c,a)$ and $cand\_class(c,a')$ coincide. $\diamond$

Theorem 5.4 Let a' be an arrow from c' to d' and let c be a subclass of c'. Let a be any arrow and let A be the real arrow represented by a. If $Isa(inher(c,a'),\ a)$ then the real arrow represented by $appr\_inher(c,a)$ is subclass of A.

Proof Let h_{min} be the *to* object of $appr\_inher(c,a)$ and let H_{min} be the real object represented by h_{min}. As $Isa(inher(c,a'),\ a)$, the object $to(a)$ is in $cand\_class(c,a')$. Therefore, $Isa(h_{min}, to(a))$ and thus, ISA$(H_{min}, to(A))$. From the fact that $Isa(inher(c,a'),\ a)$, it follows that ISA$(inher(C,A'),\ A)$ and ISA$(C, from(A))$. As $inher\_cand(C,A',H_{min})$ and $inher(C,A')$ have the same extension, it follows from Definition 2.2 (ISA) that ISA$(inher_cand(C,A',H_{min}),A)$.

References

[1] , Ait-Kaci, R. Nasr, LOGIN: A Logic Programming Language with Built-in Inheritance, *Journal of Logic Programming*, 3(3), 187-215 (1986).

[2] R. Agrawal, N.H. Gehani, Ode (Object Database and Environment): The Language and the Data Model, *Proceedings of the ACM-SIGMOD Intern. Conference on the Management of Data*, 36-45 (1989).

[3] J. Banerjee, H. Chou, J.F. Garza, W. Kim, D. Woelk, N. Ballou, H. Kim, Data Model Issues for Object-Oriented Applications, *ACM Transactions on Office Information Systems*, 5(1), 3-26 (1987).

[4] R.J. Brachman, J.G. Schmolze, An Overview of the KL-ONE Knowledge Representation System, *Cognitive Science*, 9(2), 171-216 (1985).

[5] G.H.W.M. Bronts, S.J. Brouwer, C.L.J. Martens, H.A. Proper, A Unifying Object Role Modelling Theory, *Information Systems*, 20(3), 213-235 (1995).

[6] M.J. Carey, D.J. DeWitt, S.L. Vandenberg, A Data Model and Query Language for EXODUS, *Proceedings of the ACM SIGMOD Conference on Management of Data*, 413-423 (1988).

[7] A. Cardelli, P. Wegner, On Understanding Types, Data Abstraction, and Polymorphism, *ACM Computing Surveys*, 1(4), 471-522 (1985).

[8] S. Greco, N. Leone, P. Rullo, COMPLEX: An Object-Oriented Logic Programming System, *IEEE Transactions on Knowledge and Data Engineering*, 4(4), 344-359 (1992).

[9] *Communications of the ACM*, Special Issue on Next Generation Database Systems, 34(10), (1991).

[10] P. Constantopoulos, M. Doerr, The Semantic Index System: A brief presentation, *TR Institute of Computer Science Foundation for Research and Technology-Hellas*, (http://www.ics.forth.gr/proj/isst/Systems/sis/), (1994).

[11] P. Constantopoulos, M. Doerr, Component Classification in the Software Information Base, O.Nierstrasz and D.Tsichritzis (eds.), *Object-Oriented Software Composition*, Prentice-Hall (1995).

[12] P. Constantopoulos, M. Theodorakis, Y.Tzitzikas, Developing Hypermedia Over an Information Repository, *Proceedings of the 2nd Workshop on Open Hypermedia Systems at Hypertext'96*, (1996).

[13] P. Constantopoulos, Y.Tzitzikas, Context-Driven Information Base Update, *Proceedings of the 8th Intern. Conference on Advanced Information Systems Engineering (CAiSE'96)*, 319-344 (1996).

[14] O. Deux et al. The story of O_2, *IEEE Transactions on Knowledge and Data Engineering*, 2(1), 91-108 (1990).

[15] R. Fikes, T. Kehler, The Role of Frame-Based Representation in Reasoning, *Communications of the ACM*, 28(9), 904-920 (1985).

[16] A. Heuer, P. Sander, The LIVING IN A LATTICE rule language, *Data & Knowledge Engineering*, 9, 249-286 (1993).

[17] A.H.M. ter Hofstede, Th.P. van der Weide, Expressiveness in conceptual data modelling, *Data & Knowledge Engineering*, 10(1), 65-100 (1993).

[18] R. Hull. R. King, Semantic Database Modelling, *ACM Computing Surveys*, 19(3), 202-260 (1987).

[19] M. Jarke, S. Eherer, R. Gallersdorfer, M. A. Jeusfeld, M. Staudt, ConceptBase - A Deductive Object Base Manager, *Journal of Intelligent Information Systems, Special Issue on Advances in Deductive Object-Oriented Databases*, 4(2), 167-192 (1995).

[20] M. Kifer, G. Lausen, J. Wu, Logical Foundations of Object-Oriented and Frame-Based Languages, *Journal of the Association for Computing Machinery*, 42(4), 741-843 (1995).

[21] W. Kim, Object-Oriented Databases:Definition and Research Directions, *IEEE Transactions on Knowledge and Data Engineering*, 2(3), 327-341 (1990).

[22] R. Motschnig-Pitrik, J. Mylopoulos, Classes and Instances, *International Journal of Intelligent and Cooperative Information Systems*, 1(1), 61-92 (1992).

[23] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis, Telos - a language for representing knowledge about information systems, *ACM Transactions on Information Systems*, 8(4), 325-362 (1990).

[24] J. Peckham, F. Maryanski, Semantic Data Models, *ACM Computing Surveys*, 20(3), 153-189 (1988).

[25] L. Rowe, M. Stonebraker, The POSTGRES Data Model, *Proceedings of the Intern. Conference on Very Large Data Bases (VLDB'97)*, 83-96 (1987).

[26] E. Sciore, An Extended Universal Instance Data Model, *Information Systems*, 16(1), 21-34 (1991).

[27] M. Stonebraker, G. Kemnitz, The POSTGRES Next-Generation Database Management System, *Communications of the ACM*, 34(10), 79-92, (1991).

[28] T.J. Teorey, D. Yang, J.P. Fry, A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model, *ACM Computing Surveys*, 18(2), 197-222 (1986).

[29] L.A. Zadeh, Fuzzy sets as a basis for a theory of possibility, *Fuzzy Sets and Systems*, 1, 3-28 (1978).

[30] L.A. Zadeh, Knowledge Representation in Fuzzy Logic, *IEEE Transactions on Knowledge and Data Engineering*, 1(1), 89-99 (1989).