From Relational to Object-Oriented Integrity Simplification

Manfred Jeusfeld†, **Matthias Jarke**‡

† Universität Passau, Innstraße 33, 8390 Passau, Germany
jeusfeld@andorfer.fmi.uni-passau.de

‡ RWTH Aachen, Ahornstraße 55, 5100 Aachen, Germany
jarke@picasso.informatik.rwth-aachen.de

Abstract[1]

Relational integrity checking technology can be transfered to deductive object bases by utilizing a simple logical framework for objects. The principles of object identity, aggregation and classification allow a more efficient constraint control by finer granularity of updates, composite updates and semantic constraint simplification. In many cases, meta-level constraints and deductive rules can be handled efficiently by a stepwise compilation approach. An extended integrity subsystem with these features has been implemented in the deductive object base ConceptBase.

[1] This work was supported in part by the Commission of the European Community under ESPRIT Basic Research Action 3012 (CompuLog). A version of this paper will also appear in the *Proc. Second Int. Conf. on Deductive and Object-Oriented Databases*, Munich, Dec. 1991

1. Introduction

Comprehensive and efficient integrity maintenance has been quoted as one of the major problems in next-generation databases. Systems like HiPAC [DBM88], POSTGRES [SJGP90] and DAMOKLES [REHM88] allow the developer to link procedures to certain events by so-called triggers. One problem of such systems is the correctness of the imperative procedures in the case of cascaded trigger activations. Another problem is their inflexibility concerning usage in different situations: either the triggers are activated too often or the user has to write a lot of triggers for each individual case. Experiments with the HiPAC and O_2 languages [RB90,BM91] indicate that the code to be written for user-defined triggers is at least five to ten times larger than for predicative specifications from which these triggers can be automatically generated.

Relational databases, including deductive relational databases, overcome these problems by generating specialized triggers automatically from predicative specifications of rules and constraints [DECK86, LST86, BDM88, KS88]. However, the extension of these techniques to semantic or object-oriented data models does not appear straightforward.

These models typically offer two types of constraints. Structural constraints define the meaning of abstractions such as aggregation, generalization, and classification and are hardcoded into the data model; user-defined constraints are expressed as predicative formulas. Compared with the (deductive) relational case, new problems concern the interaction of structural and explicit constraints in the data model, the role of the class concept in integrity control (seeing classes both as types and as sets of instances), the availability of metaclasses and thus of schema evolution, the efficient reconfiguration of complex objects in case of component changes, and the control of method applications.

In the context of implementing the deductive object manager ConceptBase [JARK91], we have been pursuing a research program in which we try to solve these problems systematically. Our strategy has been to reorganize successful (deductive) relational integrity checking techniques so that they fit the context of various problems encountered in object-oriented systems. This is achieved through providing a logic-based view of the object model. Our results so far indicate that this approach carries quite far, i.e., that not much additional base technology is needed to cover most "object-oriented" features. Several seemingly difficult problems turn into opportunities for further optimization that could even benefit relational systems. Specifically, this paper reports some results concerning the exploitation of the aggregation and classification abstraction found in semantic and object-oriented data models for improving the performance of deductive integrity checking.

As a basis, section 2 reviews and adapts a well-known method for integrity checking in deductive relational databases [BDM88]. Section 3 introduces a simple object framework which offers object identity, aggregation, classification, and generalization in a fairly general form, but without the automatic type inferencing techniques typical for recent work in database programming languages. This object model is part of the Telos knowledge representation language [MBJK90] used in ConceptBase and several other ongoing implementations for purposes of software information management, natural language understanding systems, hypermedia authoring environments, etc. However, the results presented in this paper are valid for fairly general sets of assumptions about object models and thus transferable to any object model that satisfies these assumptions.

The aggregation abstraction (sections 4 and 5) removes the restriction in published algorithms that the unit of change is a full tuple. On the one hand, constraint checking procedures can be attached to individual attributes. On the other, composite updates can create versions of complex objects consistently, with reduced re-checking of integrity constraints. An important application of this is version and configuration management in software databases [RJG*91,MJJG91].

The classification abstraction (sections 6 and 7) induces additional conditions for the well-formedness of explicit constraints, e.g., referential completeness. This additional check has the beneficial side-effect that ranges of variables in integrity constraints can often be reduced. Moreover, the structural constraints can be used for semantic optimization of user-defined constraints; this may substantially reduce the number of generated triggers.

A second exploitation of the classification abstraction concerns the handling of metaclasses which define the semantics of so-called slots or attribute categories. The predicative formulation of such attribute categories is often very cumbersome (we have a real-world example which contains about 25 quantifiers!) and leads to unnecessary trigger evaluation even if optimized with the usual deductive relational algorithms. For a substantial subclass of such categories (which includes, among others, the axiomatic definition of inheritance in *isa* hierarchies), a two-step compilation strategy yields a clean yet efficient solution: Delay the generation of triggered procedures until the attribute category is instantiated by a new class definition; then adapt the optimized form generated at the metalevel to this particular instance.

Our results, which concentrate on the aggregation and classification abstractions, complement the work on type inferencing (i.e., generalization hierachies) in object-oriented database languages [KLW90,ABIT90,HK87b] and KL-One-like knowledge representation languages [BBMR89]; it is our final goal to integrate these aspects in the same framework as presented here.

2. Relational Simplification Techniques

Simplification techniques for integrity constraints within relational databases have first been proposed by [NICO82]. The method was then extended to deductive databases [DECK86,BDM88]. All of these approaches base on some form of the range-restricted property [NICO82] which ensures the truth of a formula to be independent of relations not occuring in the formula. The purpose of this section is to recall the main properties of the simplification method. The theoretical background is presented in the above papers and in [JK90] where simplification is extended to deductive rules. A normal form that guarantees this property is given by the following definition (adapted from [BDM88,JK90]):

Definition 1

Let $L_1, \ldots, L_m$ be positive literals of a first-order language. Then an **integrity constraint (in range form)** is a first-order formula in miniscope and disjunctive normal form matching one of the patterns:

$$\begin{array}{l} \exists x_1, \ldots, x_n \; L_1 \wedge \ldots \wedge L_m \wedge R \\ \forall x_1, \ldots, x_n \; \neg L_1 \vee \ldots \vee \neg L_m \vee R \end{array} \tag{1}$$

Every variable $x_1, \ldots, x_n$ ($n \geq 0$) has to occur in at least one of the literals $L_1, \ldots, L_m$. The subformulas of R are either quantifier-free well-formed formulas or again in one of the above formats. If one of the variables $x_1, \ldots, x_n$ occurs in R then it is free. Variables other than quantified ones are disallowed. The literals $L_1, \ldots, L_m$ are called **range literals**.

The simplification method generates for each literal occurence L_j of an integrity constraint a **simplified form** that has to be checked when the literal is updated, i.e. inserted or deleted. Literals are interpreted by (possibly deduced) relations. In deductive databases, the simplification method must work in the presence of deductive rules. [BDM88] set up a dependency network in order to compute the derived literal updates.

Definition 2

A **deductive rule** is an integrity constraint matching

$$\forall x_1, \ldots, x_n \; \neg L_1 \vee \ldots \vee \neg L_m \vee (R \vee L_{concl}). \tag{2}$$

The formula $\forall x_1, \ldots, x_n \; \neg L_1 \vee \ldots \vee \neg L_m \vee R$ is called the **condition expression** of the deductive rule.

Let EDB be an extensional database, R be a set of deductive rules, and IC a set of integrity constraints. Then the **deductive database** (EDB,R,IC) is called **consistent** if $EDB \cup R \models \varphi$ holds for all $\varphi \in IC$.

The interpretation of formulas is classical with closed world assumption for negation. As usual, only stratified rules are considered [NT89]. In order to improve readability, we sometimes write $\varphi \Rightarrow \psi$ as an abbreviation for $\varphi^- \vee \psi$ where φ^- is the negation of φ with negation operator pushed down to the level of literals. Simplification for deductive rules is not applied to the whole formula but to its condition expression. Figure 1 shows how updates on condition literals induce updates on the conclusion literal depending on the sign. More details are given in [JK90].

Update	Sign of condition literal	Resulting update on L_{concl}
Insert	negative	Insert
Delete	negative	Delete
Insert	positive	Delete
Delete	positive	Insert

Fig. 1: Induced updates for rules

The following example demonstrates a deductive rule and an integrity constraint for patients that take drugs against symptoms they suffer from. The rule expresses that a drug helps to cure those symptoms which are effected by a component agent of the drug. The constraint prevents a patient from taking drugs that are not against any of his symptoms or that contain agents to which he is allergic.

EDB schema:

$$Patient(pname, age, takesDrug),\ \ Suffers(patient, symptom)$$
$$Drug(dname, component),\ \ Agent(aname, effectedSymptom)$$
$$Allergy(pname, aname, since)$$

R:

$$\forall\ d, s, a\ \ Drug(d, a) \wedge Agent(a, s) \Rightarrow Against(d, s) \tag{3}$$

IC:

$$\begin{aligned} \forall\ p, x, d\ \ Patient(p, x, d) \Rightarrow & (\exists s\ \ Suffers(p, s) \wedge Against(d, s)) \wedge \\ & (\forall\ a, t\ \ Drug(d, a) \Rightarrow \neg Allergy(p, a, t)) \end{aligned} \tag{4}$$

The conditions expression for rule (3) is $\forall\ d, s, a\ \neg Drug(d, a) \vee \neg Agent(a, s)$. Since the conclusion literal $Against(d, s)$ occurs only positively in constraint (4) two triggers for delete updates on the condition have to be generated (see fig. 2). The constraint contains five literals. In the corresponding range form *Suffers* and and *Against* occur positively, the rest has a negative sign. Note that variables not quantified in the simplified rules are instantiated by the update.

$\texttt{ON Delete(}Drug(d,a)\texttt{) EVALUATE } \forall s\ Agent(a,s) \Rightarrow \texttt{Delete(}Against(d,s)\texttt{)}$

$\texttt{ON Delete(}Agent(a,s)\texttt{) EVALUATE } \forall d\ Drug(d,a) \Rightarrow \texttt{Delete(}Against(d,s)\texttt{)}$

$\texttt{ON Insert(}Patient(p,x,d)\texttt{) CHECK}$
$\quad (\exists\ s\ Suffers(p,s) \wedge Against(d,s))\ \wedge (\forall\ a,t\ Drug(d,a) \Rightarrow \neg Allergy(p,a,t))$

$\texttt{ON Delete(}Suffers(p,s)\texttt{) CHECK } \forall\ x,d\ Patient(p,x,d) \Rightarrow$
$\quad (\exists\ s\ Suffers(p,s) \wedge Against(d,s))\ \wedge (\forall\ a,t\ Drug(d,a) \Rightarrow \neg Allergy(p,a,t))$

$\texttt{ON Delete(}Against(d,s)\texttt{) CHECK } \forall\ p,x\ Patient(p,x,d) \Rightarrow$
$\quad (\exists\ s\ Suffers(p,s) \wedge Against(d,s))\ \wedge (\forall\ a,t\ Drug(d,a) \Rightarrow \neg Allergy(p,a,t))$

$\texttt{ON Insert(}Drug(d,a)\texttt{) CHECK } \forall\ p,x\ Patient(p,x,d) \Rightarrow$
$\quad (\exists\ s\ Suffers(p,s) \wedge Against(d,s))\ \wedge (\forall\ t\ \neg Allergy(p,a,t))$

$\texttt{ON Insert(}Allergy(p,a,t)\texttt{) CHECK } \forall\ x,d\ Patient(p,x,d) \Rightarrow$
$\quad (\exists\ s\ Suffers(p,s) \wedge Against(d,s))\ \wedge \neg Drug(d,a)$

Fig. 2: Triggers for rule (3) and constraint (4)

The literals in predicative formulas are interpreted by relations. Thus, it is easy to detect the simplified forms affected by an update on a relation. As the simplified form for the *Patient* literal shows, relations with many attributes are advantageous with respect to the complexity of the simplified forms since they eliminate many variables (provided the variables are not governed by an existantial quantifier [BDM88]). On the other hand, relations with many attributes have an important disadvantage: updates may lead to unnecessary constraint evaluations as long as the granularity of information is fixed to tuples within relations. Changes to single components of a tuple have to be realized by updates of the whole tuple. Most algorithms found in the literature simulate updates by deletion followed by the insertion of the revised tuple. By this, unnecessary evaluation of rules and constraints can happen. If for example, the age component of a *Patient* tuple is changed, the simplified form for $\texttt{Insert}(Patient(p, x, d))$ is triggered though in fact the update cannot change the truth of the constraint. What we need is a formalism to express that a single component of a tuple or object is updated.

3. A Simple Object Model

After reviewing deductive relational integrity checking, we now introduce the object model we want to use as a framework for our extension. Deductive relational databases can be understood as first-order theories with integrity constraints [REIT84]. Analogously, deductive *object* bases can be formalized by extracting some axioms from such theories which define the structure of the object model. For example, such structural axioms may define the semantics of object identity, of abstraction principles such as aggregation, generalization, classification, and association, and of built-in system classes. The use of such axioms can be represented by introducing specialized literals into the logic that indicate, e.g., attribution, class membership, or *isa* relationships.

A number of quite sophisticated formalizations of object-oriented data models have been proposed roughly along these lines. Often inspired either by the theory of type lattices in programming languages or by KL-One-like knowledge representation formalisms, they tend to emphasize the role of object identity and generalization hierarchies with automatic classification respectively type inference [HK87,BBMR89]. There are a few logic-based formalizations of methods in object bases which specify methods by their signature. For example, [KLW90,BEER90,AG91] formalize them as functions terms, and [LV90] as predicates within a Datalog framework.

This paper deals with aspects that are complementary to type inferencing and method inheritance in that we concentrate on exploiting aggregation and classification knowledge. Generalization is seen as a logical implication between class membership predicates (see [BRAC83] and section 7). We therefore use an earlier and simpler object model that was first presented in [STAN86] and forms a subset of the knowledge representation language Telos [MBJK90]. In this section, the formalization is briefly summarized as far as it is needed later on.

Definition 3

Let ID,LAB be infinite sets of identifiers resp. labels. Then a finite subset

$$OB \subseteq \{P(oid, x, l, y) |\ oid, x, y \in ID, l \in LAB\}$$

is called a **Telos object base** iff the *oid* component is unique within OB. The elements are referred as **objects**.

It is useful to denote Telos object bases graphically as structured semantic networks. So-called **individual** objects with identical first, second and fourth component are drawn as nodes. The other objects are called **attributes** with two special cases: attributes with third component (label) *in* are called **instantiation attributes** and attributes with label *isa* are called **specialization attributes**. The *oid's* are system-generated. Figure 3 shows part of the patient example encoded as a Telos OB. The *oid's* are written as $\#Pers$, $\#takes$ etc. for the sake of readability.

$P(\#Pers,\#Pers,Person,\#Pers)$
$P(\#Pat,\#Pat,Patient,\#Pat)$
$P(\#isa1,\#Pat,isa,\#Person)$
$P(\#Drug,\#Drug,Drug,\#Drug)$
$P(\#takes,\#Pat,takes,\#Drug)$
$P(\#against,\#Drug,against,\#Sympt)$
$P(\#Sympt,\#Sympt,Symptom,\#Sympt)$
$P(\#Ag,\#Ag,Agent,\#Ag)$
$P(\#effects,\#Ag,effects,\#Sympt)$
$P(\#in1,\#QF,in,\#Drug)$
$P(\#Jack,\#Jack,Jack,\#Jack)$
$P(\#in2,\#Jack,in,\#Pat)$
$P(\#drug1,\#Jack,drug1,\#QF)$
$P(\#in3,\#drug1,in,\#takes)$
$P(\#QF,\#QF,QuasiForte,\#QF)$
$P(\#suffers,\#Pat,suffers,\#Sympt)$
$P(\#comp,\#Drug,component,\#Ag)$
$P(\#allergy,\#Pat,allergy,\#Ag)$

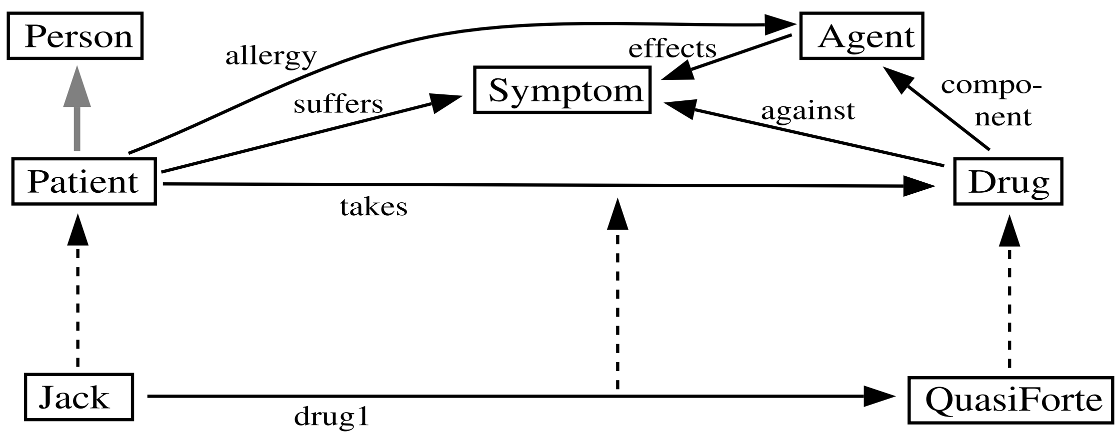

Fig. 3: Example Telos OB and its graphical representation

For objects of the form $P(oid, x, in, c)$ x is called **instance** of c (c is the **class** of x), and for $P(oid, c, isa, d)$ c is called a **subclass** of d (d is a **superclass** of c). In the following we assume that a Telos object base OB contains the classes *Object*, *Individual*, *Attribute*, *INST* and *ISA* which have as instances all objects of OB with a form indicated by the class name. The structural properties of instantiation, aggregation and specialization are defined as built-in rules and constraints. Telos classes are not distinguished from other objects. Particularly, classes are themselves instances of other (meta-) classes, metaclasses are instances of metametaclasses, etc. This infinite hierarchy is closed by a set of built-in (omega) classes [KMSB89].

Telos decomposes all information down to atomic relationships between objects. This idea was already proposed by [ABRI74] when he suggested binary relations as a data model. He also represents schema information in the same manner as instance information. Telos extends the binary relations by regarding attributes, specializations and instantiations as objects, and by certain axioms that are usual for object-oriented data models.

4. Relating Literals to Objects

The integration between this structural object model and the predicative deduction rules and integrity constraints is two-fold. From the viewpoint of the object model, predicative assertions are simply individuals with an instantiation link to a predefined metaclass called *AssertionClass*. From the viewpoint of the logical theory, well-formed formulas of the Telos assertion language must refer to the object base by using the four literals *A*, *In*, *Isa*, *P*. The objects of an object base induce solutions for the three literals:

$$\forall v, w, i \ P(i, v, in, w) \Rightarrow In(v, w) \tag{5}$$

$$\forall v, w, i \ P(i, v, isa, w) \Rightarrow Isa(v, w) \tag{6}$$

$$\forall v, w, l, a, b, l_1, o, V, W \ P(a, v, l_1, w) \wedge P(o, a, in, b) \wedge P(b, V, l, W) \Rightarrow A(v, l, w) \tag{7}$$

Additional derived instantiation and attribution links stem from the inheritance axioms. However, as we shall see in section 7, at least the relatively simple inheritance axioms used in Telos can be directly defined as metalevel assertions so that an explicit definition in this section is not necessary.

The various object-oriented data models have different granularities for the smallest possible objects. In some cases, the decomposition of an object goes down to the level of attributes. This is necessary for relating attribute updates precisely to the affected constraints. For example, in the *Patient* concept only the *takesDrug* attribute affects the constraint (4) by variable *d*. The *age* attribute has no effect. Another requirement is that classes are regarded as objects. Updates on them (e.g., the insertion of a new constraint for a class) are treated as normal database operations.

A key issue for the success of the simplification method presented in section 2 is to limit as much as possible the search space of constraints and rules to be evaluated for a given update. In the relational case, one way to achieve this is to have as many relations as possible in the schema. Then, all simplified parameterized formulas whose instantiation literals are different from those of the update are outside the search space. In that sense, the worst case is a single universal relation where only the sign of a literal and its parameter instantiation in a constraint (or rule) can prevent the system from evaluating the simplified form. Telos pays for its flexibility by falling into the worst case. We have a single "base literal" P and the three derived literals *In(v,w)*, *Isa(v,w)* and *A(v,l,w)*. If their formulas (5-7) were treated as deductive rules, each update of a P literal would potentially trigger a simplified instance of each of the three rules. Only non-matching parameters could then prevent rule evaluation. This situation is unacceptable since all transactions contain object updates. Even worse, applying stratification to only four literals would severely limit the expressiveness of deductive rules. In the following, the concept of *concerned classes* is developed that will almost solve the both problems.

Object-oriented languages offer the notion of classes to define structure within the OB. This observation motivates the decision to restrict the formulas to a many-sorted first-order language [LT85] where the variables are bound to classes:

$$\exists\ x/C\ \varphi \qquad \text{stands for} \qquad \exists x\ In(x,C) \wedge \varphi \tag{8}$$

$$\forall\ x/C\ \varphi \qquad \text{stands for} \qquad \forall x\ \neg In(x,C) \vee \varphi \tag{9}$$

The typed quantifications guarantee that the formulas are range-restricted and are used to determine the so-called concerned class of a literal occurence.

Definition 4

Let (OB,R,IC) be a deductive object base, $\varphi \in R \cup IC$, and L be a literal occurence in φ. An object cc of OB is called a **concerned class** of L (w.r.t. φ) if for all possible instantiations L' of L in φ:

If $OB \cup R \models L'$ then $OB \cup R \models \exists x\ In(x,cc)$.

If $OB \cup R \models \neg L'$ then $OB \cup R \models \exists x\ \neg In(x,cc)$.

In other words, the truth of $In(x,cc)$ is necessary for the truth of L'. Because of the variable typing (8,9) and the definition of the three Telos literals (5-7) the set of concerned classes is never empty: the class *Object* always applies. To get a finer range the object bases are restricted by following two constraints.

Referential integrity

Any referenced object must exist:

$$\begin{aligned} &\forall\ oid,x,l,y\ P(oid,x,l,y) \Rightarrow \\ &\qquad\qquad \exists\ x_1,l_x,y_1,x_2,l_y,y_2\ P(x,x_1,l_x,y_1) \wedge P(y,x_2,l_y,y_2) \end{aligned} \tag{10}$$

With referential integrity and the definition of the literals it follows that the class $INST$ is a range for the In literal: the object i in (5) may never be part of a Telos OB if v or w are missing. Thus, $In(v,w)$ succeeds exactly for the instances of class $INST$. Analogously, ISA can be taken as the concerned class of Isa literals and *Object* for the P literal of Telos. If the second argument in $In(v,w)$ is a constant then this object w is a concerned class, too.

The interesting case is the A literal due to the three condition literals and its numerous quantifications in (7). If all variables are free in such a literal then *Attribute* is the smallest concerned class. Since attributes make up a major portion of a Telos object base this is not satisfactory. A second restriction enables A literals to be connected to "user defined" attribute categories such as *#takes* in the example OB.

Uniqueness property

For any literal $A(x,l,y)$ occuring in a formula φ the label l must be instantiated and the number of solutions aid fulfilling

$$\exists\ aid, c, Y\ In(x,c) \wedge P(aid,c,l,Y) \tag{11}$$

must be exactly one.

This assumption enforces the existence of a unique attribute in the OB that corresponds to the literal. Then, aid is a concerned class. In order to validate stratification, the name of the literals in the formulas have to be replaced by the names of the concerned classes:

For $In(x,c)$ write $c(x)$ if c is a constant, otherwise write $INST(x,c)$. $Isa(c,d)$ is replaced by $ISA(c,d)$. Thirdly, $A(x,l,c)$ becomes $aid(x,y)$ where aid is the concerned class.

Instead of four we now have a multitude of different literals, one for each class and attribute of a class. This solves the problem of stratification mentioned before. It should be noted that the original literal can be obtained very efficiently from the instantiation to the concerned class. For $In(x,c)$ and $Isa(c,d)$ this is obvious. For the $A(x,l,y)$ literal one only has to fetch the object $P(a,x,l',y)$ for a given instantiation $In(a,aid)$ to the concerned class. A minor problem remains with literals having *INST* or *ISA* as concerned classes since they belong to the lowest stratification level [LST86] due to the rules (5-7). As a consequence, deductive rules with such conclusion literals are fairly restricted. Section 7 will provide a method for handling a lot of such cases.

Continuing the patient example, the rule and the constraint now appear as typed formulas:

$$\begin{aligned}&\forall\ d/Drug,\ s/Symptom,\ a/Agent\\ &\quad A(d,component,a) \wedge A(a,effects,s) \Rightarrow A(d,against,s)\end{aligned} \tag{12}$$

$$\begin{aligned}&\forall\ p/Patient,\ d/Drug\\ &\quad A(p,takes,d) \Rightarrow (\exists\ s/Symptom\ A(p,suffers,s) \wedge A(d,against,s)) \wedge\\ &\quad (\forall\ a/Agent\ A(d,component,a) \Rightarrow \neg A(p,allergy,a))\end{aligned} \tag{13}$$

Only the required attributes appear in the formulations. The age attribute, for example, that caused problems in formula (4) has now disappeared. In the range form, the sorts of the variables are represented by instantiation literals. The labels of the individual objects *Drug* etc. are replaced with their identifiers. In an implementation, this step can automatically be carried out in the compilation phase. The rule and constraint have to be read as:

$$\begin{aligned}&\forall\ d,s,a\ In(d,\#Drug) \wedge In(s,\#Sympt) \wedge In(a,\#Ag) \wedge\\ &\quad A(d,component,a) \wedge A(a,effects,s) \Rightarrow A(d,against,s)\end{aligned} \tag{14}$$

$$\begin{aligned}&\forall\ p,d\ In(p,\#Pat) \wedge In(d,\#Drug) \wedge A(p,takes,d) \Rightarrow\\ &\quad (\exists\ s\ In(s,\#Sympt) \wedge\ A(p,suffers,s) \wedge A(d,against,s)) \wedge\\ &\quad (\forall\ a\ In(a,\#Ag) \wedge\ A(d,component,a) \Rightarrow \neg A(p,allergy,a))\end{aligned} \tag{15}$$

With the object base of fig. 3, the smallest concerned classes for the In literals are $\#Drug$, $\#Sympt$, $\#Ag$, $\#Pat$ and for the A literals $\#comp$, $\#effects$, $\#against$, $\#takes$, $\#suffers$, $\#allergy$. Five simplified forms are generated for the rule as shown in fig. 4. From the nine simplified forms for constraint (15) only the first and last is shown in the figure.

The close relation between literals and concerned classes allows to determine the updated literals and the affected simplified forms in a single step. Furthermore, the more subclasses are defined and used in predicative formulas, the smaller is the number of instances to be expected for the concerned classes. Implementations of Telos (and of many other object-oriented databases, e.g. Iris [LWH90]) promote the idea of extensibility through updates to the set of classes in the database. Thus, poorly balanced class hierachies can be corrected at run-time of the system to improve performance of integrity checking.

ON Delete($In(d, \#Drug)$) EVALUATE $\forall\ s, a\ In(s, \#Sympt) \wedge In(a, \#Ag) \wedge$
$A(d, component, a) \wedge A(a, effects, s)$ $\Rightarrow$ Delete($A(d, against, s)$)
ON Delete($In(s, \#Sympt)$) EVALUATE $\forall\ d, a\ In(d, \#Drug) \wedge In(a, \#Ag) \wedge$
$A(d, component, a) \wedge A(a, effects, s)$ $\Rightarrow$ Delete($A(d, against, s)$)
ON Delete($In(a, \#Ag)$) EVALUATE $\forall\ d, s\ In(d, \#Drug) \wedge In(s, \#Sympt) \wedge$
$A(d, component, a) \wedge A(a, effects, s)$ $\Rightarrow$ Delete($A(d, against, s)$)
ON Delete($A(d, component, a)$) EVALUATE $\forall\ s\ In(d, \#Drug) \wedge In(s, \#Sympt) \wedge$
$In(a, \#Ag) \wedge A(a, effects, s)$ $\Rightarrow$ Delete($A(d, against, s)$)
ON Delete($A(a, effects, s)$) EVALUATE $\forall\ d\ In(s, \#Sympt) \wedge In(a, \#Ag) \wedge$
$In(a, \#Ag) \wedge A(d, component, a)$ $\Rightarrow$ Delete($A(d, against, s)$)

ON Insert($In(p, \#Pat)$) CHECK $\forall\ d\ In(d, \#Drug) \wedge A(p, takes, d) \Rightarrow$
$(\exists\ s\ In(s, \#Sympt) \wedge A(p, suffers, s) \wedge A(d, against, s)) \wedge$
$(\forall\ a\ In(a, \#Ag) \wedge A(d, component, a) \Rightarrow \neg A(p, allergy, a))$
...
ON Insert($A(p, allergy, a)$) CHECK $\forall\ d\ In(p, \#Pat) \wedge In(d, \#Drug) \wedge$
$A(p, takes, d) \Rightarrow (\exists\ s\ In(s, \#Sympt) \wedge A(p, suffers, s) \wedge A(d, against, s)) \wedge$
$(\neg In(a, \#Ag) \vee \neg A(d, component, a))$

Fig. 4: Triggers for rule (14) and constraint (15)

Compared to the relational method in section 2 we have the advantage that updates to irrelevant attributes do not trigger any evaluation. However, the adapted method has two disadvantages: concurrent updates to attributes of the same object each trigger a formula evaluation, and there are a lot of "unnecessary" triggers for the In literals. The next two sections present solutions for these problems.

5. Support of Composite Updates

So far, the adaption of the simplification method to the aggregation/decomposition principle of object-oriented data models provided the finer granularity for single attribute updates. The method falls short if more than one literal is updated in one transaction: if a constraint contains multiple attributes of the same object then the method would generate simplified forms for each single attribute update and thus re-evaluate common subexpressions.

This weakness can be compensated by aggregating A literals around a common source object that are updated in "one step", i.e., during the same transaction. **Update rules** define a literal U to be equivalent to the conjunction of several A literals:

$$\forall x, y_1, \ldots, y_n \; U(x, y_1, \ldots, y_n) \Leftrightarrow A(x, l_1, y_1) \wedge \ldots \wedge A(x, l_n, y_n) \tag{16}$$

All the literals $A(x, l_i, y_i)$ have to be *single-valued*, i.e., the number of solutions for a given x and l_i must be less or equal one. Let φ be a constraint that contains an expression $A(x, l_1, y_1) \wedge \ldots \wedge A(x, l_n, y_n)$. If one replaces the conjunction by $U(x, y_1, \ldots, y_n)$ one gets a formula φ' which can now be simplified for the literal U. If an update contains U then it is sufficient to check this simplified form instead of the n simplified forms for $A(x, l_1, y_1)$,...,$A(x, l_n, y_n)$. The correctness and completeness follows from the restriction to single-valued attributes. The mapping of $U(x, y_1, \ldots, y_n)$ to objects of the OB is described by the update rule and by the definition of the A literal (7).

As an example, consider the formula

$$\begin{aligned} \forall\; e, m, x, y \;\; & A(e, boss, m) \wedge A(e, salary, x) \wedge \\ & A(m, salary, y) \Rightarrow A(x, lessequal, y) \end{aligned} \tag{17}$$

Suppose we have the update rule $\forall e, m, x \; U_1(e, m, x) \Leftrightarrow A(e, boss, m) \wedge A(e, salary, x)$, i.e., *boss* and *salary* of an employee are updated in one step. Then formula (17) can be expressed by

$$\forall\; e, m, x, y \;\; U_1(e, m, x) \wedge A(m, salary, y) \Rightarrow A(x, lessequal, y) \tag{18}$$

The simplified form for U_1 is

`ON Insert(`U_1*(e,m,x)*`) CHECK` $\forall\; y \;\; A(m, salary, y) \Rightarrow A(x, lessequal, y)$

which eliminates three quantifiers. The success of introducing updates rules depends on the number of constraints that contain a matching conjunction, and on the frequency of updates that contain such an update literal U. In any case, the notion of composite updates prevents the object-oriented method from being worse than the relational one: introduce an update rule for each relation and possibly drop irrelevant attributes. Composite updates do not bury the advantages of the simplification based on single attribute updates. Such updates can still be checked by their simplified forms.

6. Semantic Optimization Based on Structural Axioms

It is a characteristic of object-oriented databases that information is organized into class hierachies. The Telos predicative sublanguage reflects this feature by typed quantifications. One advantage is that all formulas are by definition range-restricted but as a disadvantage we have additional *In* literals interpreting the variable types (see section 4). In the basic algorithm, these literals would be subject to generation of simplified forms. This section proposes a method for eliminating a considerable subset of such literals by exploiting structural integrity constraints found in most object-oriented data models and the strong relationship between literals and classes stated in section 4.

The following structural integrity constraint restricts the instantiation of classes (a kind of type checking): If an object id is an instance of a class c then the source x of id must be instance of the source X of c and its destination y must be instance of the destination Y of c.

Instantiation axiom

$$\begin{aligned}\forall\ id, c\ In(id, c) \Rightarrow \exists x, l_1, y, X, l, Y\ \ & P(id, x, l_1, y) \wedge P(c, X, l, Y) \wedge \\ & In(x, X) \wedge In(y, Y)\end{aligned} \tag{19}$$

Due to the uniqueness of the oid component of objects (def. 3) the existentially quantified variables are uniquely determined by id or c, resp.

Lemma 1

Let (OB,R,IC) fulfill the uniqueness property and the instantiation axiom, and let $A(x, l, y)$ be a literal occuring in a formula $\varphi \in R \cup IC$. Let further cc be a concerned class of the literal and $P(cc, X, l, Y) \in OB$. Then each occurence of

$$A(x, l, y) \wedge In(x, X)$$
$$A(x, l, y) \wedge In(y, Y)$$

in φ can be replaced by $A(x, l, y)$ without changing the truth of φ.

The proof follows from the the definition of the A literal and from the instantiation axiom. Deductive rules having $A(x, l, y)$ as their conclusion create no problem since the conclusion literal can only contribute to a condition literal $A(x', l, y')$ if the concerned classes are the same.

As an example, we show how lemma 1 improves the optimization of rule (14). All three *In* literals can be eliminated: $In(d, \#Drug))$ and $In(a, \#Agt)$ appear in conjunction with $A(d, component, a)$. The concerned class of this literal is $P(\#comp, \#Drug, component, \#Ag)$ which makes the lemma applicable. The eliminations of the remaining $In(s, \#Sympt)$ and the four *In* literals in constraint (15) follow analogously. The two formulas shrink to:

$$\forall\ d, s, a\ A(d, component, a) \wedge A(a, effects, s) \Rightarrow A(d, against, s) \tag{20}$$

$$\begin{aligned}\forall\ p, d\ A(p, takes, d) \Rightarrow & (\exists\ s\ A(p, suffers, s) \wedge A(d, against, s)) \wedge \\ & (\forall\ a\ A(d, component, a) \Rightarrow \neg A(p, allergy, a))\end{aligned} \tag{21}$$

$$\texttt{ON Delete(}A(d, component, a)\texttt{) EVALUATE } \forall s\ A(a, effects, s) \Rightarrow \texttt{Delete(}A(d, against, s)\texttt{)}$$

$$\texttt{ON Delete(}A(a, effects, s)\texttt{) EVALUATE } \forall\ d\ A(d, component, a) \Rightarrow \texttt{Delete(}A(d, against, s)\texttt{)}$$

...

$$\texttt{ON Insert(}A(p, allergy, a)\texttt{) CHECK } \forall\ d\ A(p, takes, d) \Rightarrow (\exists\ s\ A(p, suffers, s) \wedge A(d, against, s)) \wedge \neg A(d, component, a)$$

Fig. 5: Triggers for (20) and (21)

Figure 5 demonstrates the effect of the optimization method. Compared to fig. 4, the simplified forms can be evaluated more efficiently due to the smaller number of literals. Even more important, no triggers have to be generated for the eliminated literals: the number is reduced from five to two for the rule, and from nine to five for the constraint. This is the same number as observed in fig. 2 for the relational model. But remember: each class and each attribute constitute a new literal. Thereby, the trigger search space for object bases is significantly better partitioned than for relational databases.

7. Stepwise Compilation of Meta-Level Information

A few object-oriented databases regard classes themselves as objects that can be updated like any other object. Common properties of these classes are represented in so-called meta classes. The attributes of such meta classes have the role of attribute categories. A typical example is the *necessary*[2] attribute category [KMSB89]: each token that is instance of a class with an attribute of category *necessary* must instantiate this attribute:

$$\begin{aligned}\forall\ A, X, l, Y, x\ \ & In(A, \#necessary) \wedge P(A, X, l, Y) \wedge In(x, X) \\ & \Rightarrow \exists\ a, l_1, y\ P(a, x, l_1, y) \wedge In(a, A)\end{aligned} \tag{22}$$

There are two literals $In(x, X)$ and $In(a, A)$ that have a "bad" concerned class: $INST$. Such literals are called **meta literals**. If we simplify the formula directly then we have to generate simplified forms triggered by every insertion (deletion) of any instantiation relationship. That is prohibitively expensive. The following lemma shows a possibility to overcome this problem for formulas of a certain form.

[2] The semantic database SIM [JGF*88] has a similar category called *required.*

Lemma 2

> Let OB be a Telos object base and φ be an integrity constraint in range form containing a meta literal whose second argument is an universally quantified variable c_i not governed by an existantial quantifier. Let L_j be a range literal containing c_i. Then OB fulfills φ iff OB fulfills all simplified forms generated for all solutions for L_j that are true for OB.

The lemma is a corollary to the soundness and completeness of the basic simplification method [BDM88]. The simplified forms for the solutions of L_j contain at least one universal quantifier less than the original constraint. Therefore, iterative application of lemma 2 always terminates. The range form guarantees the existence of at least one range literal L_j. The choice of the "best" L_j depends on the number of solutions for L_j (which are always finite since OB is finite) and the number of variables in L_j. As an example we apply the method to the *necessary* attribute category. We use $In(A, \#necessary)$ as a choice for the range literal L_j. For the solution $In(\#against, \#necessary)$ where $P(\#against, \#Drug, against, \#Sympt)$ is part of the OB we obtain:

$$\forall x\ In(x, \#Drug) \Rightarrow \exists\ a, l_1, y\ P(a, x, l_1, y) \wedge In(a, \#against) \tag{23}$$

Due to the key property the only possible solution for $P(\#against, X, l, Y)$ is $P(\#against, \#Drug, against, \#Sympt)$. For the same reason one can omit simplification for the literal $P(a, x, l_1, y)$. As result, the deduced integrity constraint must be checked only if a new instance of *#Drug* is inserted or an instance of the attribute *#against* is deleted.

As a second example we apply the technique to a deductive rule that describes the **inheritance of class membership** (in fact, another structural axiom of Telos): each instance of a subclass is also instance of its superclasses.

$$\forall\ x, c, d\ \ In(x, c) \wedge Isa(c, d) \Rightarrow In(x, d) \tag{24}$$

This rule has two meta literals. We take $Isa(c, d)$ as the range literal to simplify it. Let $Isa(\#Pat, \#Pers)$ be a new solution. The corresponding simplified deductive rule is:

$$\forall x\ \ In(x, \#Pat) \Rightarrow In(x, \#Pers) \tag{25}$$

with the condition expression $\forall x\ \neg In(x, \#Pat)$. According to the table in section 2, two simplified forms for the insertion and deletion of $In(x, \#Pat)$ have to be generated. It follows that evaluation of the simplified form for an insertion on $In(x, \#Pat)$ directly deduces $In(x, \#Pers)$ without any further access to the object base! Analogously, deletions of $In(x, \#Pat)$ directly induce deletions of $In(x, \#Pers)$.

8. Conclusions

The aim of this paper was not to propose a new integrity checking method. Instead, technology from the relational database area was transferred to the object-oriented domain. It turned out that characteristic features of object-oriented data models like aggregation, classification and meta classes are not obstacles but new opportunities to increase efficiency and expressiveness of the relational simplification method.

We have implemented the method in the KBMS ConceptBase. Algorithms and the maintenance of integrity constraints as part of the object base are reported in [JK90]. We see the following contributions:

- By decomposing complex objects into atomic attribute relationships the simplification of predicative formulas is supported down to the level of attribute updates. This prevents unnecessary evaluations which are common in the relational method. The same degree of granularity is supported by the CACTIS system [HK87] but it is used there to maintain derived attributes by more procedural attributed graphs.
- The finer degree of granularity is accompanied by the support of composite updates. As a consequence, the object-oriented simplification method is in no case worse than the relational one.
- The many-sorted first order language automatically guarantees range-restrictedness of the formulas. By exploitation of an instantiation axiom one can eliminate most of the type literals from the formula. Experiments with ConceptBase show a speed increase by a factor of 2 to 5 compared to formulas optimized with the basic relational techniques for the kinds of examples shown in this paper.
- Deductive rules and integrity constraints located at the metaclass level can be handled efficiently by an incremental compilation approach. Examples are the inheritance axioms and the *necesssary* attribute category. This feature improves expressiveness of the formulas and enables true schema updates.

In hindsight, these results may not be all that surprising: One of the reasons of introducing semantic and object-oriented databases was to include more semantics in the object structure, thus reducing the need for defining explicit constraints. Nevertheless, it is gratifying to observe that this idea can actually be operationalized, as our experiments seem to indicate.

Several important problems and opportunities addressed in ongoing work are not covered in this paper. The first concerns *complex objects*. Object bases are supposed to deliver exactly the data structures the application program asks for. In order to preserve data independence view mechanisms are used to describe the complex objects [SS91]. Such views can be formulated as queries that are deductive rules with a many-argument conclusion [STAU90]. Some results on how integrity checking for updates on a complex object can be efficiently realized in an abductive framework are given in [MJJG91]. On the other hand, the rule triggering method sketched in section 2 offers a way for an automated complex object update induced by updates in the component objects.

Many object bases use hand-written *methods* for checking the correctness of certain updates. They can profit from our results if they automatically generate the correctness checking triggers from integrity constraints (this has already been suggested by [BM91] for the O_2 system). Furthermore, if methods are refined from specifications which describe the method's updates then the corresponding integrity checking procedures can be automatically included into the code [JMW*90].

Acknowledgements. We would like to thank Eva Krüger who considerably contributed to the deductive integrity checking component of ConceptBase [KRÜG89]. Many thanks also to Hendrik Decker and our collegues in the CompuLog project for help and discussions.

9. References

[ABIT90] Abiteboul,S. (1990). Towards a deductive object-oriented database language. *Data & Knowledge Engineering 5*, 1990.

[ABRI74] Abrial,J.R. (1974). Data semantics. In Klimbie and Koffeman (eds.): Data Base Management, North-Holland Publ.

[AG91] Abiteboul,S., Grumbach,S. (1991). A rule-based language with functions and sets. *ACM Transactions on Database Systems* 16(1), March 1991.

[BBMR89] Borgida,A., Brachman,R.J., McGuinness,D.L., Resnick,L.A. (1989). CLASSIC: a structural data model for objects. *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, Portland, Oregon.

[BDM88] Bry,F., Decker,H., Manthey,R. (1988). A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. *Proc. EDBT'88*, Venice, Italy.

[BEER90] Beeri,C. (1990). A formal approach to object-oriented databases. *Data & Knowledge Engineering* 5, 1990.

[BM91] Bouzeghoub, M., Metais, E. (1991). Semantic modeling of object-oriented databases. *Proc. 17th Int. Conf. on Very Large Data Bases*, Barcelona.

[BRAC83] Brachman,R.J. (1983). What IS-A is and isn't: an analysis of taxonomic links in semantic networks. *IEEE Computer* 16(10), Oct. 1983.

[DBM88] Dayal,U., Buchmann,A., McCarthy,D. (1988). Rules are objects too: a knowledge model for an active object-oriented database system. *Proc. 2nd Int. Workshop on Object-Oriented Database Systems*, Bad Münster, Germany.

[DECK86] Decker,H. (1986). Integrity enforcement on deductive databases. *Proc. First Int. Conf. on Expert Database Systems*, Menlo Park, Calif.

[HK87] Hudson,S.E., King,R. (1987). Object-oriented database support for software environments. *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, San Francisco, Calif.

[HK87b] Hull,R., King,R. (1987). Semantic database modeling: survey, applications, and research issues. *ACM Computing Surveys* 19(3), Sept. 1987.

[JARK91] Jarke,M. (ed.,1991). ConceptBase V3.0 user manual. Report MIP-9106, Universität Passau.

[JGF*88] Jagannathan,D., Guck,R.L., Fritchman,B.L., Thompson,J.P., Tolbert,D.M. (1988). SIM: a database system based on the semantic data model. *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, Chicago, Ill.

[JJR89] Jarke,M., Jeusfeld,M., Rose,T. (1989). Software process modeling as a strategy for KBMS implementation. *Proc. First Int. Conf. on Deductive and Object-Oriented Databases*, Kyoto, Japan.

[JK90] Jeusfeld,M., Krüger,E. (1990). Deductive integrity maintenance in an object-oriented setting. Report MIP-9013, Universität Passau, Germany, 1990.

[JMW*90] Jeusfeld,M., Mertikas,M., Wetzel,I., Jarke,M., Schmidt,J.W. (1990). Database application development as an object modeling activity. *Proc. 16th Int. Conf. on Very Large Data Bases*, Brisbane, Australia.

[KLW90] Kifer,M., Lausen,G., Wu,J. (1990). Logical foundations of object-oriented and frame-based languages. *Reihe Informatik* 3/1990, Universität Mannheim.

[KMSB89] Koubarakis,M., Mylopoulos,J., Stanley, M., Borgida,A. (1989). Telos: features and formalization. Technical Report KR-89-04, University of Toronto, Ont.

[KRÜG89] Krüger,E. (1989). Integritätsprüfung in deduktiven Objektbanken am Beispiel von ConceptBase. Diploma thesis, Universität Passau, Germany.

[KS88] Kowalski,R., Sadri,F. (1988). A theorem-proving approach to database integrity. In Minker (ed.): Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann Publishers.

[LWH90] Lyngbæk,P., Wilkinson,K., Hasan,W. (1990). The Iris kernel architecture. *Proc. EDBT'90*, Venice, Italy.

[LST86] Lloyd,J.W., Sonnenberg,E.A., Topor,R.W. (1986). Integrity constraint checking in stratified databases. Technical Report 86/5, Department of Computer Science, University of Melbourne.

[LT85] Lloyd,J.W., Topor,R.W. (1985). A basis for deductive database systems. *J. Logic Programming* 2, 1985.

[LV90] Laenens,E., Vermeir,D. (1990). A fixpoint semantics for ordered logic. *J. Logic Computat.* 1(2), 1990.

[MBJK90] Mylopoulos,J., Borgida,A., Jarke,M., Koubarakis,M. (1990). Telos: a language for representing knowledge about information systems. *ACM Trans. Information Systems* 8(4).

[MJJG91] Miethsam,A., Jarke,M., Jeusfeld,M., Gocek,M. (1991). Towards a logic-based reconstruction of software configuration management. Report, ESPRIT Basic Reseach Action 3012 (CompuLog), RWTH Aachen, July 1991.

[NICO82] Nicolas,J.-M. (1982). Logic for improving integrity checking in relational databases. *Acta Informatica* 18(3), Dec. 1982.

[NT89] Naqvi,S., Tsur,S. (1989). A logical language for data and knowledge bases. Computer Science Press.

[RB90] Rios-Zertuche, D., Buchmann, A. (1990). Execution models for active databases: a comparison. Technical Report, GTE Laboratories, Waltham, Mass., 1990.

[REHM88] Rehm,S. et al. (1988). Support for design processes in a structurally object-oriented database system. *Proc. 2nd Int. Workshop on Object-Oriented Database Systems*, Bad Münster, Germany.

[REIT84] Reiter,R. (1984). Towards a logical reconstruction of relational database theory. In Brodie et al. (eds.): On Conceptual Modelling, Springer, 1984.

[RJG*91] Rose,T., Jarke,M., Gocek,M., Maltzahn,C., Nissen,H. (1991). A decision-based configuration process environment. *Software Engineering Journal*, Special Issue on Software Process and its Support, to appear.

[SJGP90] Stonebraker,M., Jhingran,A., Goh,J., Potiamos,S. (1990). On rules, procedures, caching and views in database systems. *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, Atlantic City, NJ.

[SS91] Scholl,M., Schek,H.-J. (1991). Supporting views in object-oriented databases. *Data Engineering Bulletin*, June 1991.

[STAU90] Staudt,M. (1990). Anfragerepräsentation und -auswertung in deduktiven Objektbanken. Diploma thesis, Universität Passau, Germany.

[STAN86] Stanley,M.T. (1986). CML: a knowledge representation language with application to requirements modeling. M.S. thesis, University of Toronto, Ont.