Integrity Enforcement in Object-Oriented Databases*

Klaus–Dieter Schewe[1], Bernhard Thalheim[2],
Joachim W. Schmidt[1], Ingrid Wetzel[1]

[1] University of Hamburg, Dept. of Computer Science,
Vogt-Kölln-Str. 30, D-W-2000 Hamburg 54, FRG
[2] University of Rostock, Dept. of Computer Science,
Albert-Einstein-Str. 21, D-O-2500 Rostock, FRG

Abstract. In contrast to the relational model methods in OODBs must enforce structurally defined constraints such as inclusion and referential constraints. It has been shown that this is possible for basic *generic update operations* that are determined by the schema. However, such operations only exist for value-representable classes.
In this paper we generalize this result and show that integrity enforcement is always possible. Given some arbitrary method S and some static or transition constraint $\mathcal{I}$ there exists a *greatest consistent specialization* (GCS) $S_{\mathcal{I}}$ of S with respect to $\mathcal{I}$. Such a GCS behaves nice in that it is compatible with the conjunction of constraints, inheritance and refinement.
Moreover, it is possible to derive simple representations of GCSs for basic update operations with respect to distinguished classes of explicitly stated static constraints. For the GCS construction of a user-defined operation, however, it is in general not sufficient to replace the involved primitive update operations by their GCSs.

1 Introduction

Consistency is a crucial property of database application systems. In general a database may be considered as a triplet $(\mathcal{S}, \mathcal{O}, \mathcal{C})$, where $\mathcal{S}$ defines a structure, $\mathcal{O}$ denotes a collection of state changing operations and $\mathcal{C}$ is a set of constraints. Constraints can be classified into static, transition and general dynamic constraints describing legal states, state transitions or state sequences respectively. Then the *consistency problem* is to guarantee that each specified operation $o \in \mathcal{O}$ will never violate any constraint $\mathcal{I} \in \mathcal{C}$. *Integrity enforcement* aims at the derivation of a new set $\mathcal{O}'$ of operations such that $(\mathcal{S}, \mathcal{O}', \mathcal{C})$ satisfies this property.

In this paper we address the integrity enforcement problem with respect to static and transition constraints. Our analysis will be based on the object oriented datamodel (OODM) introduced in [22] which is based on a clear distinction between *values* and *objects* as required e.g. in [6, 7]. In this model an object o consists of a unique identifier id, a set of (type-, value-)pairs (T_i, v_i),

* This work has been supported in part by research grants from the E.E.C. Basic Research Action 3070 FIDE: "Formally Integrated Data Environments".

a set of (reference-, object-)pairs (ref_j, o_j) and a set of methods $meth_k$. A type is defined algebraically by constructors, selectors, functions and axioms similar to [8, 18, 10, 12].

The class concept provides the grouping of objects having the same structure which uniformly combines aspects of object values and references. The extent of classes varies over time, whereas types are immutable. Relationships between classes are represented by references together with referential constraints on the object identifiers involved. Moreover, each class is accompanied by a collection of methods. A schema is given by a collection of class definitions together with explicit constraints. This datamodel is related to some other work on the foundations of object oriented databases [1, 7, 11, 15, 25], but it is distinct from existing systems [4, 2, 14]. The description of the OODM will appear in Section 3.

In general, verification techniques based on predicate transformers are applicable [20]. This includes to show that each update operation starting in a legal state with respect to the static constraints can only end up in legal states. This also comprises that methods on subclasses overriding inherited methods must be specializations. The latter proof obligation can be exploited to formalize the satisfaction of transition constraints, since each transition constraint can be regarded as a general predicatively specified method. Then a method satisfies a transition constraint iff it specializes the method associated with the constraint. In Section 2 we describe in part the theoretical background of consistency verification.

An obvious disadvantage of the verification approach is that it does not help the user in writing consistent operations. An alternative is to generate a *Greatest Consistent Specialization* (GCS) of a given method with respect to the given constraints. This comprises the following problems:

(i) Does a GCS exist in general? Is it compatible with the conjunction of constraints, optimization, inheritance and refinement?

(ii) How does a GCS look like in the OODM with respect to distinguished classes of constraints?

(iii) How to enforce integrity of general user-defined operations with respect to arbitrary constraints? Is it sufficient to replace involved primitive operations by their GCSs?

In Section 4 we address these problems for static constraints. We show the existence of GCSs and also discuss compatibility results with respect to the conjunction of constraints, specialization and refinement. These positive result set up a fundamental distinction to trigger approaches [13], where conjunctivity can not be guaranteed. In Section 5 we describe the structure of GCSs in the OODM with respect to specific classes of constraints. Moreover, we show that the general enforcement problem can not be reduced to primitive operations. In Section 6 we derive the existence of GCSs for transition constraints and also discuss compatibility properties. We conclude in Section 7 discussing some open problems concerning the handling of general dynamic constraints and the structure of GCSs in general.

2 Theoretical Background

For the moment let us abstract from the specific database context. Look at a database being defined as some *state space* X with typed state variables $x_1 :: T_1, \ldots, x_n :: T_n$. Then *state transitions* on X are expressible by (partial, nondeterministic) guarded commands with a sophisticated axiomatic semantics defined by predicate transformers. Formulae in first-order logic can be used to express both static and transition integrity *constraints* on X. We adopt this approach in order to formalize OODM as well as the integrity enforcement problem within a strict mathematical framework that will later be applied to the OODM.

2.1 Types

In general a type is specified by a collection of constructors, selectors and other functions – the signature of the type – and axioms defined by universal Horn formulae [23]. Now let N_P, N_T, N_F, and V denote arbitrary pairwise disjoint, countably infinite sets representing a reservoir of parameter-, type-, function-, and variable-names respectively.

Definition 1. A *type signature* Σ consists of a type name $t \in N_T$, a finite set of supertype-/function-pairs $T \subseteq N_T \times N_F$, a finite set of parameters $P \subseteq N_P$, a finite set of base types $B \subseteq N_T$ and pairwise disjoint finite sets C, S, $F \subseteq N_F$ of constructors, selectors and functions such that there exist predefined arities $ar(c) \in (P \cup B^* \cup \{t\})^* \times \{t\}$, $ar(s) \in \{t\} \times (P \cup B^* \cup \{t\})$ and $ar(f) \in (P \cup B^* \cup \{t\})^* \times (P \cup B^* \cup \{t\})$ for each $c \in C$, $s \in S$ and $f \in F$.

We write $f : t \rightarrow t'$ to denote a *supertype-/function-pair* $(t', f) \in T$. We write $c : t_1 \times \ldots \times t_n \rightarrow t$ to denote a *constructor* of arity $(t_1 \ldots t_n, t)$, $s : t \rightarrow t'$ to denote a *selector* of arity (t, t') and $f : t_1 \times \ldots \times t_n \rightarrow t'$ to denote a *function* of arity $(t_1 \ldots t_n, t')$. If $t_i = b_i^0 \ldots b_i^m \in B^*$, we write $b_i^0(b_i^1 \ldots b_i^m)$. We call $\mathcal{S} = P \cup B \cup \{t\}$ the set of *sorts* of the signature Σ.

Definition 2. A *type declaration* consists of a type signature Σ with type name t such that there exists a type declaration for each $b \in B - \{t\}$ and a set Ax of Horn formulae over Σ. Moreover, if $b_i^0(b_i^1 \ldots b_i^m)$ with $b_i^j \in B$ occurs within a constructor, selector or function, then b_i^0 must have been declared as a parameterized type with m parameters. We say that (Σ, Ax) defines the *parameterized type* $t(\alpha_1, \ldots, \alpha_n)$, iff $P = \{\alpha_1, \ldots, \alpha_n\} \neq \emptyset$ or the *proper type* t respectively.
A *type* t is defined either by a type declaration or by mutually recursive equations involving t as a variable.

The semantics of a type is given by term generated algebras that are quotients of the term algebra defined by the constructors. Subtyping is modelled by the use of a continous function taking the subtype to the supertype. Recursive types are fixpoints of functors. It can be shown that even the guarded commands give rise to a type $GC(\alpha, \beta, \gamma)$, where α (β) is the type of the input (output) and γ is the type of the underlying state space. See [20] for more details.

2.2 State Transitions

In general non-deterministic partial state transitions S on a state space X can be described by a subset of $\mathcal{D} \times \mathcal{D}_\perp$, where $\mathcal{D}$ denotes the set of possible states on X and $\mathcal{D}_\perp = \mathcal{D} \cup \{\perp\}$, where $\perp$ is a special symbol used to indicate non-termination. It can be shown that this is equivalent to defining two predicate transformers $wp(S)$ and $wlp(S)$ associated with S satisfying the pairing condition $wp(S)(\mathcal{R}) \Leftrightarrow wlp(S)(\mathcal{R}) \wedge wp(S)(true)$ and the universal conjunctivity of $wlp(S)$. They assign to some postcondition $\mathcal{R}$ the *weakest (liberal) precondition* of S to establish $\mathcal{R}$. Informally these conditions can be characterized as follows:

- $wlp(S)(\mathcal{R})$ characterizes those initial states such that all terminating executions of S will reach a final state characterized by $\mathcal{R}$ provided S is defined in that initial state, and
- $wp(S)(\mathcal{R})$ characterizes those initial states such that all executions of S terminate and will reach a final state characterized by $\mathcal{R}$ provided S is defined.

Such operations S can be specified by guarded commands in the style of Dijkstra [9, 17]:

Definition 3. Let X be some state space. A *guarded command* S on X consists of a name S, a set of input-parameters $\{\iota_1, \ldots, \iota_k\}$, a set of output-parameters $\{o_1, \ldots, o_l\}$ and a body. To each input-parameter ι_i corresponds a type T_i and to each output-parameter o_j corresponds a type O_j. The *body* of S is recursively built from the following constructs:

(i) assignment $x := E$, where x is a state variable in X or a local variable within S and E is a term of the same type as x,
(ii) $skip$, $fail$, $loop$,
(iii) sequential composition $S_1; S_2$, choice $S_1 \Box S_2$, projection $x :: T \mid S$, guard $\mathcal{P} \rightarrow S$, restricted choice $S_1 \otimes S_2$, where $\mathcal{P}$ is a well-formed formula and x is a variable of type T, and
(iv) instantiation $x'_1, \ldots, x'_i \leftarrow S'(E'_1, \ldots, E'_j)$, where S' is the name of another operation ($S = S'$ is possible) on X with input-parameters $\iota'_1, \ldots, \iota'_j$ and output-parameters $o'_1, \ldots, o'_i$, such that the variables o'_f, x'_f have the same type and the term E'_g has the same type as the variable ι'_g.

Each variable occurring in the S has a well-defined scope. The scoping rules are omitted. Furthermore, we omit the detailed definition of the predicate transformers $wlp(S)$ and $wp(S)$ [17, 20]. We only give an informal description of the less usual operations *projection*, *guard* and *restricted choice*. Projection gives the introduction of a new local variable x of the given type. A guard $\mathcal{P} \rightarrow S$ gives a precondition $\mathcal{P}$ for S. If $\mathcal{P}$ is not satisfied, the whole operation is undefined. Restricted choice $S \otimes T$ means to execute S unless it is undefined in which case T is taken. The basic commands $skip$, $fail$, $loop$ are only introduced for theoretical completeness: $skip$ does nothing, $fail$ is always undefined, and $loop$ never terminates.

Here we dispense with any structuring of state spaces into modules. However, in order to define "extended operations" we need to know for each operation S the subspace $Y \subseteq X$ such that S does neither read nor change the values in $X - Y$. In this case we call S a Y-operation on X. We omit the formal details [17, 20].

2.3 Consistency Proof Obligations

General constraints and arbitrary operations on a state space X raise the problem whether consistency as defined by the constraints is always satisfied by the operations. One approach to address this problem is to use general verification techniques. The verification approach consists in the derivation (and proof) of general proof obligations expressed in the predicate transformer calculus.

The static constraints on a state space X, i.e. first-order formulae $\mathcal{I}$ with $fr(\mathcal{I}) \subseteq X$ partition the state space, i.e. the collection of the mutable classes into two distinguished subspaces. States not satisfying the constraints should never be reached.

In general inherited operations can be overwritten. Unless inheritance is simply regarded as a copying mechanism we should ensure that this can be done in a concise way, i.e., overriding should be restricted to "specialization". The intuition behind this definition is that whenever an execution of the specialized operation T establishes some post-predicate $\mathcal{R}$, then this execution should already be one of the general method S.

Transition constraints on X are expressible as first-order formulae $\mathcal{J}$ with $fr(\mathcal{J}) \subseteq X \cup X'$, where X' is a disjoint copy of X. We may then exploit a weak equivalence between guarded commands and predicative specifications. We may associate with the transition constraint $\mathcal{J}$ the guarded command $\Phi(\mathcal{J}) = x' \mid \mathcal{J} \rightarrow x := x'$ where x (x') is used as an abbreviation for the collection $x_1, \ldots, x_n$ ($x'_1, \ldots, x'_n$) of (state) variables. Satisfying $\mathcal{J}$ is equivalent to each operation S specializing $\Phi(\mathcal{J})$. Hence the following formal definitions:

Definition 4. Let X be a state space, $Z \subseteq Y \subseteq X$ subspaces, $\mathcal{I}$ a static and $\mathcal{J}$ a transition constraint on X, S a Z-operation and T a Y-operation.

(i) S is *consistent* with respect to $\mathcal{I}$ iff $\mathcal{I} \Rightarrow wlp(S)(\mathcal{I})$ holds on X.
(ii) T *specializes* S iff $wp(S)(true) \Rightarrow wp(T)(true)$ and $wlp(S)(\mathcal{R}) \Rightarrow wlp(T)(\mathcal{R})$ hold for all Z-predicates $\mathcal{R}$ (denoted $T \sqsubseteq S$).
(iii) S is *consistent* with respect to $\mathcal{J}$ iff $\mathcal{R}\ wlp(\Phi(\mathcal{J}))(\mathcal{R}) \Rightarrow wlp(S)(\mathcal{R})$ holds for all X-predicates.

Note that $\sqsubseteq$ defines a partial order on operations. There exists an equivalent characterization of transition consistency that avoids the quantification over all state predicates [19]. See [20] for more details.

3 Fundamentals of Object Oriented Data Modelling

In the following assume to be given a type system $\mathbb{T}$ as described e.g. in Section 2.1. Basically such a type system consists of some *basic types* such as $BOOL$, $\mathbb{N}$, $\mathbb{Z}$, $STRING$, etc., *type constructors* (parameterized types) for record, finite sets, lists, etc. and a *subtyping* relation. Moreover, assume that (mutually) *recursive types*, i.e. types defined by (a system of) domain equations, exist in $\mathbb{T}$. As an alternative to our definition of $\mathbb{T}$ in Section 2.1 we may restrict $\mathbb{T}$ being one of the type systems defined in [3]. In addition we suppose the existence of an abstract identifier type ID in $\mathbb{T}$ without any non-trivial supertype. Arbitrary *types* can then be defined by nesting. A type T without occurrence of ID will be called a *value-type*.

Now let $N_P, N_T, N_C, N_R, N_F, N_M$ and V denote arbitrary pairwise disjoint, denumerable sets representing parameter-, type-, class-, reference-, function-, method- and variable-names respectively.

3.1 The Concept of a Class

The OODM in [22] distinguishes between values grouped into types and objects grouped into classes. The extent of classes varies over time, whereas types are immutable. Relationships between classes are represented by references together with referential constraints on the object identifiers involved. Moreover, each class is accompanied by a collection of methods defined by deterministic guarded commands [17, 19, 20].

Each object in a class consists of an identifier, a collection of values and references to objects in other classes. Identifiers can be represented using the unique identifier type ID. Values and references can be combined into a representation type, where each occurence of ID denotes references to some other classes. Therefore, we may define the structure of a class using parameterized types.

Definition 5. (i) Let t be a value type with parameters $\alpha_1, \ldots, \alpha_n$. For distinct reference names $r_1, \ldots, r_n \in N_R$ and class names $C_1, \ldots, C_n \in N_C$ the expression derived from t by replacing each α_i in t by $r_i : C_i$ for $i = 1, \ldots, n$ is called a *structure expression*.

(ii) A *class* consists of a class name $C \in N_C$, a structure expression S, a set of class names $D_1, \ldots, D_m \in N_C$ (in the following called the set of *superclasses*), a set of static constraints $\mathcal{I}_1, \ldots, \mathcal{I}_k$ and a set of transition constraints $\mathcal{J}_1, \ldots, \mathcal{J}_l$. We call r_i the *reference* named r_i from class C to class C_i. The type derived from S by replacing each reference $r_i : C_i$ by the type ID is called the *representation type* T_C of the class C.

(iii) A *schema* $\mathcal{S}$ is a finite collection of classes $C_1, \ldots, C_n$ closed under references and superclasses together with a collection of static constraints $\mathcal{I}_1, \ldots, \mathcal{I}_n$ and a collection of transition constraints $\mathcal{J}_1, \ldots, \mathcal{J}_m$.

(iv) An *instance* $\mathcal{D}$ of a schema $\mathcal{S}$ assigns to each class C a value $\mathcal{D}(C)$ of type $PFUN(ID, T_C)$ such that all implicit and explicit constraints on $\mathcal{S}$ are satisfied.

Here we dispense with giving a concrete syntax for constraints and methods. On the representation level (see Section 3.2) we may use deterministic guarded commands for methods and first-order formulae for constraints. Distinguished classes of static constraints will be introduced in Section 5.1. Subclasses inherit the methods of their superclasses, but overriding is allowed as long as the new method is a specialization of all its corresponding methods in its superclasses.

3.2 The Representation of a Schema

We now associate with each schema $\mathcal{S}$ a state space X such that each class C in $\mathcal{S}$ is represented by a state variable $x_C :: PFUN(ID, T_C)$ in X called the *class type* of C. $PFUN(\alpha, \beta)$ is the type constructor for partial function from α to β with finite domain. Moreover, C gives rise to *referential constraints* defined by the structure S and *class inclusion constraints* defined by the set of superclasses of C. All other constraints on $\mathcal{S}$ and C are directly translatable in constraints

on X. Methods on C are Y-operations on X with $Y = \{x_C\}$. Note that due to identifiers being only an internal concept, only value-defined operations, i.e. all input- and output-types are value types, are accessible to the user. Let us now formally describe the form of structurally defined inclusion and referential constraints.

Definition 6. Let C, C' be classes with representation types T_C and T'_C respectively and let $o : T_C \times ID \to BOOL$ be a function.

(i) If T_C be a subtype of T'_C via $f : T_C \to T'_C$, a *class inclusion constraint* on C and C' is a constraint in the form

$$\forall i :: ID. \forall v :: T_C . member(Pair(i, v), x_C) = true \Rightarrow member(Pair(i, f(v)), x_{C'}) = true \ , \quad (1)$$

where $Pair$ is the constructor of $PAIR(\alpha, \beta)$ and $member$ is a function on finite sets $FSET(\alpha)$, hence also on the subtype $PFUN(\alpha, \beta)$.
In the general case a *class inclusion constraint* on C and C' has the form

$$\forall i :: ID. member(i, dom(x_C)) = true \Rightarrow member(i, dom(x_{C'})) = true \ .$$

(ii) A *referential constraint* on C and C' is a constraint in the form

$$\forall i, j :: ID. \forall v :: T_C . member(Pair(i, v), x_C) = true \wedge o(v, j) = true \Rightarrow member(j, dom(x_{C'})) = true \ . \quad (2)$$

It is easy to see that each class D in the set of superclasses of C gives rise to an inclusion constraint. Moreover, each reference $r : E$ occurring in the structure expression S of C gives rise to a referential constraint with the function o determined by the type underlying S. Then $o(v, j) = true$ means that the identifier j occurs within v at a place corresponding to the reference.

3.3 Value Representability

Let us first concentrate on primitive update methods, i.e. insertion, deletion and update of a single object on a classes C. In contrast to the relational datamodel such update operations can not always be derived in the object-oriented case, because the abstract identifiers have to be hidden from the user. However, in [21, 22] it has been shown that for *value-representable* classes these operations are uniquely determined by the schema and consistent with respect to the implicit referential and inclusion constraints.

Definition 7. A class C in a schema $\mathcal{S}$ with representation type T_C is called *value-representable* iff there exists a proper value type V_C such that for all instances $\mathcal{D}$ of $\mathcal{S}$ there is a function $c : T_C \to V_C$ such that for $\mathcal{D}$

(i) the condition $\forall i, j :: ID. \forall v, w :: T_C . member(Pair(i, v), x_C) = true \wedge member(Pair(j, w), x_C) = true \ \wedge \ c(v) = c(w) \ \Rightarrow \ i = j$ holds and

(ii) for each pair (V'_C, c') consisting of a proper value type V'_C and a function $c' : T_C \to V'_C$ such that property (i) is satisfied there exists a function $c'' : V_C \to V'_C$ that is unique on $c(codom_{\mathcal{D}}(x_C))$ with $c' = c'' \circ c$.

If only the first property holds, C is called *value-identifiable*.

Theorem 8. *Let C be a class in a schema $\mathcal{S}$. Then there exist unique canonical update operations on C iff C and all its superclasses are value-representable.*

For details and proofs see [22].

3.4 An Example

Let us now finalize the presentation of the datamodel by a simple example.

Example 1. Assume the existence of a value type PERSON defined elsewhere. A class C named PERSONC may be defined as follows. Some details such as the definition of the method named "exists" are omitted, but we remark that the described insert-method is in fact the canonical one [22].

PERSONC ==
  Structure PAIR(PERSON , spouse : PERSONC)
  Constraints $\forall$ I,J :: ID . $\forall$ V :: T_C . member(Pair(I,V),x_C) = true $\wedge$
    member(Pair(J,V),x_C) = true $\Rightarrow$ I = J
  Methods
    insert(P :: V_C = PAIR(PERSON,V_C)) == I $\leftarrow$ insert$'$(P)
  (**hidden**)
  I :: ID $\leftarrow$ insert$'$(P :: $\overline{V_C}$ = PAIR(PERSON,UNION($\overline{V_C}$,ID))) ==
    B :: BOOL | (B $\leftarrow$ exists(P,x_C) ;
      B = true $\rightarrow$
        (I :: ID | (member(I,dom(x_C)) = false $\rightarrow$
            P$''$ :: T_C |
              ($\exists$ P$'$ :: T_C . P$'$ = P $\rightarrow$ P$''$:= P $\otimes$
                (J :: ID | J $\leftarrow$ insert$'$(substitute(P,I,second(P))) ;
                  P$''$:= Pair(first(P),J))) ;
          x_C := union(x_C,single(Pair(I,P$''$)))))
        $\otimes$ skip
End PERSONC □

4 Enforcing Static Integrity

An alternative to consistency verification is the computation of methods that enforce all constraints of a schema. We now address this problem first for static constraints and generalize Theorem 8. Our approach starts with a formalization of the integrity enforcement problem focussing on GCSs. We show that GCSs always exist and are unique (up to semantic equivalence). On this formal basis we are able to describe certain compatibility results and outline the structure of GCSs with respect to basic update operations and distinguished classes of static constraints.

4.1 The Problem

Suppose now to be given an update operation S and a static constraint $\mathcal{I}$. Assume that S is an X-operation, whereas $\mathcal{I}$ is defined on Y with $X \subseteq Y$. The idea is to construct a "new" Y-operation $S_{\mathcal{I}}$ that is consistent with respect to $\mathcal{I}$ and can be used to replace S. Roughly speaking this means that the effect of $S_{\mathcal{I}}$ on the state variables in X should not be different from the effect of S. Formally this is expressed by the specialization relation introduced in Definition 4. Clearly, if any there will be more than one such specialization. Hence the idea to distinguish one of them as the "greatest", i.e. all others should specialize it.

Before giving now the definition of a GCS let us first remark that for any predicate transformer f the *conjugate predicate transformer* f^* is defined by $f^*(\mathcal{R}) = \neg f(\neg\mathcal{R})$. Hence the following definition of a *greatest consistent specialization*:

Definition 9. Let $X \subseteq Y$ be state spaces, S an X-operation and $\mathcal{I}$ a static integrity constraint on Y. A Y-operation $S_\mathcal{I}$ is a *Greatest Consistent Specialization* (GCS) of S with respect to $\mathcal{I}$ iff

(i) $wlp(S)(\mathcal{R}) \Rightarrow wlp(S_\mathcal{I})(\mathcal{R})$ holds on Y for all formulae $\mathcal{R}$ with $fr(\mathcal{R}) \subseteq X$,
(ii) $wp(S(true) \Rightarrow wp(S)_\mathcal{I})(true)$ holds on Y,
(iii) $\mathcal{I} \Rightarrow wlp(S_\mathcal{I})(\mathcal{I})$ holds on Y and
(iv) for each Y-operation T satisfying properties (i) – (iii) (instead of $S_\mathcal{I}$) we have

(a) $wlp(S_\mathcal{I})(\mathcal{R}) \Rightarrow wlp(T)(\mathcal{R})$ for all formulae $\mathcal{R}$ with $fr(\mathcal{R}) \subseteq X$ and
(b) $wp(S_\mathcal{I})(true) \Rightarrow wp(T)(true)$.

Note that properties (i) and (ii) require $S_\mathcal{I}$ to be a specialization of S. Property (iii) requires $S_\mathcal{I}$ to be consistent with respect to the constraint $\mathcal{I}$. Finally, property (iv) states that each consistent specialization T of S with $T \sqsubseteq S$ also specializes $S_\mathcal{I}$.

Based on the formal definition of a GCS we can now raise the following questions:

- Does such a GCS always exist? If it does, is it uniquely determined by S and $\mathcal{I}$ (up to semantic equivalence)?
- Is the GCS $S_\mathcal{I}$ of a deterministic operation S itself deterministic? If not, how to achieve a deterministic consistent operation?
- Does a GCS $(S_{\mathcal{I}_1})_{\mathcal{I}_2}$ (provided it exists) with respect to more than one integrity constraint depend on the order of enforcement? Is it sufficient to take $(S_{\mathcal{I}_1})_{\mathcal{I}_2}$ in order to enforce integrity with respect to $\mathcal{I}_1 \wedge \mathcal{I}_2$?
- What is the relation between integrity enforcement and inheritance, i.e. is the GCS $T_\mathcal{I}$ of a specialization T of S a specialization of the GCS $S_\mathcal{I}$ of S?
- How does a GCS look like (if it exists)?

Partial results for these questions will be discussed in the next subsections.

4.2 Greatest Consistent Specializations

Let us first address the existence problem. Based on the axiomatic semantics via predicate transformers we can show that a GCS always exists and is uniquely determined up to semantic equivalence. First, however, we need some general properties concerning the specialization order $\sqsubseteq$.

Lemma 10. *Let $\mathcal{T}$ be a set of X-operations. Then there exists the least upper bound $S_0 = \bigsqcup_{S \in \mathcal{T}} S$ with respect to $\sqsubseteq$. S_0 is uniquely determined (up to semantic equivalence) and satisfies*

$$(i) \qquad wlp(S_0)(\mathcal{R}) \Leftrightarrow \bigwedge_{S \in \mathcal{T}} wlp(S)(\mathcal{R}) \text{ and} \tag{3}$$

(ii) $\quad wp(S_0)(\mathcal{R}) \;\Leftrightarrow\; \bigwedge_{S \in \mathcal{T}} wp(S)(\mathcal{R})$ (4)

for all formulae $\mathcal{R}$ *with* $fr(\mathcal{R}) \subseteq X$.

Proof. Let S_0 be defined (up to semantic equivalence) by (3) and (4). Then the universal conjunctivity and the pairing condition are trivially satisfied. It follows that S_0 is an X-operation.

Let $T \in \mathcal{T}$ and $\mathcal{R}$ be an arbitrary formula with $fr(\mathcal{R}) \subseteq X$. Then we have:

- $\bigwedge_{S \in \mathcal{T}} wlp(S)(\mathcal{R}) \;\Rightarrow\; wlp(T)(\mathcal{R})$ and
- $\bigwedge_{S \in \mathcal{T}} wp(S)(\mathcal{R}) \;\Rightarrow\; wp(T)(\mathcal{R})$.

Thus, S_0 is an upper bound of $\mathcal{T}$. If T_0 is any upper bound of $\mathcal{T}$, we get

- $wlp(T_0)(\mathcal{R}) \;\Rightarrow\; wlp(T)(\mathcal{R})$ and
- $wp(T_0)(\mathcal{R}) \;\Rightarrow\; wp(T)(\mathcal{R})$

for all $T \in \mathcal{T}$, hence also

- $wlp(T_0)(\mathcal{R}) \;\Rightarrow\; \bigwedge_{S \in \mathcal{T}} wlp(S)(\mathcal{R})$ and
- $wp(T_0)(\mathcal{R}) \;\Rightarrow\; \bigwedge_{S \in \mathcal{T}} wp(S)(\mathcal{R})$.

It follows that $S_0 \sqsubseteq T_0$, i.e. S_0 is the least upper bound. □

Note that for $\mathcal{T} = \emptyset$ the least upper bound is $fail$. Now we are prepared to present our result concerning the unique existence of a GCS.

Theorem 11. *Let S be an X-operation, $X \subseteq Y$ and $\mathcal{I}$ a static integrity constraint on Y. Then there exists a greatest consistent specialization $S_{\mathcal{I}}$ of S with respect to $\mathcal{I}$. Moreover, $S_{\mathcal{I}}$ is uniquely determined (up to semantic equivalence) by S and $\mathcal{I}$.*

Proof. Let $\mathcal{T}$ be the set of Y-operations T satisfying (in place of $S_{\mathcal{I}}$ the properties (i) – (iii) of Definition 9 and let $S_{\mathcal{I}} = \bigsqcup_{T \in \mathcal{T}} T$. By definition of a least upper bound $S_{\mathcal{I}}$ satisfies properties (i), (ii) and (iv). Property (iii) follows from Lemma 10(i). □

Although Theorem 11 states that there is always a (unique) solution of the integrity enforcement problem, it does not help us in constructing a GCS, since its proof is non-constructive. Another general problem is that there are usually more than just one static integrity constraint. If we successively build GCSs, can we guarantee that the final result will be independent of the order of constraints? Is the final result the same, if we simply take the conjunction of all constraints? We address these two problems next.

Theorem 12. *Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be static constraints on Y_1 and Y_2 respectively. If for any operation S the GCS with respect to $\mathcal{I}_i$ is denoted by $S_{\mathcal{I}_i}$ $(i = 1, 2)$, then for any X-command with $X \subseteq Y_1 \cap Y_2$ the GCSs $(S_{\mathcal{I}_1})_{\mathcal{I}_2}$ and $(S_{\mathcal{I}_2})_{\mathcal{I}_1}$ are semantically equivalent.*

Proof. For symmetric reasons it is sufficient to show

$(S_{\mathcal{I}_1})_{\mathcal{I}_2} \sqsubseteq (S_{\mathcal{I}_2})_{\mathcal{I}_1}$.

We have $\mathcal{I}_2 \Rightarrow wlp((S_{\mathcal{I}_1})_{\mathcal{I}_2})(\mathcal{I}_2)$. Because of the transitivity of the implication $(S_{\mathcal{I}_1})_{\mathcal{I}_2}$ then satisfies properties (i) – (iii) with respect to S and $\mathcal{I}_2$, hence by property (iv) we get $(S_{\mathcal{I}_1})_{\mathcal{I}_2} \sqsubseteq S_{\mathcal{I}_2}$. We have

$$\mathcal{I}_1 \Rightarrow wlp(S_{\mathcal{I}_1})(\mathcal{I}_1) \Rightarrow wlp((S_{\mathcal{I}_1})_{\mathcal{I}_2})(\mathcal{I}_1) \ .$$

The first implication is the consistency of $S_{\mathcal{I}_1}$ with respect to $\mathcal{I}_1$, the second implication follows from property (i) for $(S_{\mathcal{I}_1})_{\mathcal{I}_2}$ with $\mathcal{R} = \mathcal{I}_1$.

Thus, $(S_{\mathcal{I}_1})_{\mathcal{I}_2}$ satisfies properties (i) –(iii) with respect to $S_{\mathcal{I}_2}$ and $\mathcal{I}_1$, i.e.

$(S_{\mathcal{I}_1})_{\mathcal{I}_2} \sqsubseteq (S_{\mathcal{I}_2})_{\mathcal{I}_1}$ by property (iv).

□

Theorem 13. *Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be static constraints on Y_1 and Y_2 respectively. If for any operation S the GCS with respect to $\mathcal{I}_i$ is denoted by $S_{\mathcal{I}_i}$ $(i = 1, 2)$, then for any X-command with $X \subseteq Y_1 \cap Y_2$ the GCSs $(S_{\mathcal{I}_1})_{\mathcal{I}_2}$ and $S_{(\mathcal{I}_1 \wedge \mathcal{I}_2)}$ coincide on initial states satisfying $\mathcal{I}_1 \wedge \mathcal{I}_2$, i.e., $\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow (S_{\mathcal{I}_1})_{\mathcal{I}_2}$ and $\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{(\mathcal{I}_1 \wedge \mathcal{I}_2)}$ are semantically equivalent.*

Proof. From the transitivity of the implication and Definition 9 it follows that

$(S_{\mathcal{I}_1})_{\mathcal{I}_2} \sqsubseteq S_{\mathcal{I}_1 \wedge \mathcal{I}_2}$ holds.

Then also

$\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow (S_{\mathcal{I}_1})_{\mathcal{I}_2} \sqsubseteq \mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2}$ holds.

We have to show the converse. Let

$$\bar{S}_{\mathcal{I}_1} = \mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2} \otimes S_{\mathcal{I}_1} \ .$$

Then we get $wlp(S)(\mathcal{R}) \Rightarrow wlp(\bar{S}_{\mathcal{I}_1})(\mathcal{R})$ and $wp(S)(\mathcal{R}) \Rightarrow wp(\bar{S}_{\mathcal{I}_1})(\mathcal{R})$ for all formulae $\mathcal{R}$ with $fr(\mathcal{R}) \subseteq X$.

Moreover, $\mathcal{I}_1 \Rightarrow wlp(\bar{S}_{\mathcal{I}_1})(\mathcal{I}_1)$ holds, hence by Definition 9 it follows that $\bar{S}_{\mathcal{I}_1} \sqsubseteq S_{\mathcal{I}_1}$.

If we define

$$\bar{S}_{\mathcal{I}_1,\mathcal{I}_2} = \mathcal{I}_2 \rightarrow \bar{S}_{\mathcal{I}_1} \otimes (S_{\mathcal{I}_1})_{\mathcal{I}_2} \ ,$$

then by analogous arguments we derive $\bar{S}_{\mathcal{I}_1,\mathcal{I}_2} \sqsubseteq (S_{\mathcal{I}_1})_{\mathcal{I}_2}$. In particular we conclude

$$\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow \bar{S}_{\mathcal{I}_1,\mathcal{I}_2} \sqsubseteq \mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow (S_{\mathcal{I}_1})_{\mathcal{I}_2} \ ,$$

but $\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow \bar{S}_{\mathcal{I}_1,\mathcal{I}_2}$ is semantically equivalent to $\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow \bar{S}_{\mathcal{I}_1}$ and this to $\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow (S_{\mathcal{I}_1})_{\mathcal{I}_2}$. Hence the theorem. □

In Section 3.1 we required methods on subclasses that override inherited methods to be specializations. Since we regard methods as operations on some state space defined by the schema, the problem occurs, whether the GCS of a specialized operation T of S remains to be a specialization of the GCS of S. Fortunately this is also true.

Theorem 14. *Let $S_{\mathcal{I}}$ be the GCS of the X-operation S with respect to the static integrity constraint $\mathcal{I}$ defined on Y with $X \subseteq Y$. Let T be a Z-operation that specializes S. If $\mathcal{I}$ is regarded as a constraint on $Y \cup Z$, then the GCS $T_{\mathcal{I}}$ of T with respect to $\mathcal{I}$ is a specialization of $S_{\mathcal{I}}$.*

Proof. From the transitivity of the implication and the consistency of $T_{\mathcal{I}}$ with respect to $\mathcal{I}$ it follows that $T_{\mathcal{I}}$ satisfies properties (i) – (iii) of Definition 9, hence $T_{\mathcal{I}} \sqsubseteq S_{\mathcal{I}}$. □

In Section 3.1 methods were introduced as deterministic guarded commands. Hence the question whether this property is preserved under GCS construction. Unfortunately this is not true as the following example shows.

Example 2. Let x, y and z be state variables, all of type $FSETS(T)$, where $FSETS(\alpha)$ is the finite set constructor (see e.g. [20]) and T is any value type. Let $X = \{y\}$ and $Y = \{x, y, z\}$. Then we define an X-operation S by

$$S(t :: T) \;==\; y' :: FSET(T) \;|\; y = Union(y', Single(t)) \wedge member(t, y') = false \;\rightarrow\; y := y'$$

and a static constraint $\mathcal{I}$ on Y by

$$x = Union(y, z) \;\wedge\; \forall t :: T.member(t, y) = true \Rightarrow member(t, z) = false \;\;.$$

Then the GCS $S_{\mathcal{I}}$ has the form

$$S_{\mathcal{I}}(t :: T) \;==\; \mathcal{I} \;\rightarrow S(t)\;;\; (\,z := Union(z, Single(t)) \;\square\; x' FSET(T) \;|\; x = Union(x', Single(t)) \wedge member(t, x') = false \;\rightarrow\; x \;:=\; x'\,) \;\otimes\; S(t)\;\;.$$

We omit the formal proof of the properties of Definition 9. □

A general approach to remove non-determinism is *operational refinement* as defined in [20]. However, operational refinement allows to "complete" a specification of an operation S whenever S is undefined are never terminating. In this paper we do not regard completion via refinement. Therefore, we regard the notion of specialization instead. Due to [20, Proposition 4.1] it is easy to see that whenever S is consistent with respect to a static constraint $\mathcal{I}$, then each specialization T of S does so, too.

Hence a deterministic specialization of $S_{\mathcal{I}}$ is still a consistent specialization of S with respect to $\mathcal{I}$. The next theorem states that we can choose a maximal one.

Theorem 15. *Let $S_{\mathcal{I}}$ be the GCS of a deterministic operation S with respect to the static constraint $\mathcal{I}$ and let T be some deterministic specialization of $S_{\mathcal{I}}$. Then T specializes S. Moreover, if T' is any deterministic specialization of S that is consistent with respect to $\mathcal{I}$, then T' also specializes some deterministic specialization T of $S_{\mathcal{I}}$.*

Proof. The first statement is trivial, since $\sqsubseteq$ is a partial order.

If T' is a deterministic specialization of S consistent with respect to $\mathcal{I}$, then according to Definition 9 T' must be a specialization of $S_{\mathcal{I}}$, hence the second statement. □

5 Enforcing Static Integrity in the OODM

Let us now apply the results of Section 4.2 to the OODM presented in Section 3. In this section we restrict ourselves to the basic update operations of Theorem 8 and describe the structure of their GCSs with respect to distinguished classes of static constraints. Moreover, we discuss whether the general problem of GCS construction could be reduced to these basic operations. Unfortunately this is not true.

5.1 Distinguished Classes of Static Constraints

Let us now introduce some kinds of explicit static constraints are generalizations of constraints known from the relational model, e.g. functional and key constraints, general inclusion and exclusion constraints, multi-valued dependencies and path constraints.

Definition 16. Let C, C_1, C_2 be classes in a schema $\mathcal{S}$ and let $c^i : T_C \to T_i$ $(i = 1, 2, 3)$ and $c_i : T_{C_i} \to T$ $(i = 1, 2)$ be functions.

(i) A *functional constraint* on C is a constraint of the form
$$\forall i, i' :: ID. \forall v, v' :: T_C . c^1(v) = c^1(v') \wedge member(Pair(i, v), x_C) = true \wedge member(Pair(i', v'), x_C) = true \Rightarrow c^2(v) = c^2(v') \ . \quad (5)$$
A functional constraint is called a *value constraint* iff neither T_1 nor T_2 contains ID.

(ii) A *uniqueness constraint* on C is a constraint of the form
$$\forall i, i' :: ID. \forall v, v' :: T_C . c^1(v) = c^1(v') \wedge member(Pair(i, v), x_C) = true \wedge member(Pair(i', v'), x_C) = true \Rightarrow i = i' \ . \quad (6)$$
A uniqueness constraint on C is called *trivial* iff $T_C = T_1$ and $c^1 = id$ hold.

(iii) A *general inclusion constraint* on C_1 and C_2 is a constraint of the form
$$\forall t :: T. \exists i_1 :: ID, v_1 :: T_{C_1} . member(Pair(i_1, v_1), x_{C_1}) = true \wedge c_1(v_1) = t \Rightarrow \exists i_2 :: ID, v_2 :: T_{C_2} . member(Pair(i_2, v_2), x_{C_2}) = true \wedge c_2(v_2) = t. \quad (7)$$

(iv) An *exclusion constraint* on C_1, C_2 is a constraint of the form
$$\forall i_1, i_2 :: ID. \forall v_1 :: T_{C_1}. \forall v_2 :: T_{C_2} . member(Pair(i_1, v_1), x_{C_1}) = true \wedge member(Pair(i_2, v_2), x_{C_2}) = true \Rightarrow c_1(v_1) \neq c_2(v_2) \ . \quad (8)$$

(v) An *object generating constraint* on C is a constraint of the form
$$\forall i_1, i_2 :: ID. \forall v_1, v_2 :: T_C . member(Pair(i_1, v_1), x_C) = true \wedge member(Pair(i_2, v_2), x_C) = true \wedge c^1(v_1) = c^1(v_2) \Rightarrow \exists i :: ID, v :: T_C . member(Pair(i, v), x_C) = true \wedge c^1(v) = c^1(v_1) \wedge c^2(v) = c^2(v_1) \wedge c^3(v) = c^3(v_2) \ . \quad (9)$$

Note that the definition of uniqueness constraints is a generalization of the key concept and object generating constraints are a straightforward generalization of multi-valued dependencies in the relational model [5, 24]. The following definition extends these constraints to path constraints.

Definition 17. (i) Let $C_1, \ldots, C_n$be classes in a schema $\mathcal{S}$ with representation types $T_{C_1}, \ldots, T_{C_n}$ and let referential constraints on C_{i-1}, C_i be defined via $o_i : T_{C_{i-1}} \times ID \to BOOL$. Then $C_1, \ldots, C_n$ define a *path* in $\mathcal{S}$ and the corresponding *path expression* is given by
$$member(Pair(i_1, v_1), x_{C_1}) = true \wedge o_2(v_1, i_2) = true \wedge member(Pair(i_2, v_2), x_{C_2}) = true \wedge \ldots \wedge o_n(v_{n-1}, i_n) = true \wedge member(Pair(i_n, v_n), x_{C_n}) = true \ . \quad (10)$$

(ii) Let C, C' be classes in a schema $\mathcal{S}$ and let $\mathcal{P}$ be a { (general) inclusion | exclusion | functional | uniqueness | object generating } constraint on C, C' or C respectively. If $C_1, \ldots, C_n$ and $C'_1, \ldots, C'_m$ are paths in $\mathcal{S}$ with $C_n = C$ and $C'_m = C'$, then replacing the corresponding path expressions for $member(Pair(i, v), x_C) = true$ and $member(Pair(i', v'), x_{C'}) = true$ respectively in $\mathcal{P}$ defines a *path constraint* $\mathcal{P}'$ on C_1 and C'_1. We assume all free variables in $\mathcal{P}'$ other than x_C and $x_{C'}$ to be universally quantified. More precisely we call $\mathcal{P}'$ a { *(general) path inclusion* | *path exclusion* | *path functional* | *path uniqueness* | *path object generating* } *constraint.*

5.2 Transforming Static Constraints into Primitive Operations

Let us now try to generalize the result of theorem 8 with respect to explicit static constraints. Let $\mathcal{S}$ be some schema and let $\mathcal{I}$ be an explicit static constraint on $\mathcal{S}$. We want to derive again insert-, delete- and update-operations for each class C in $\mathcal{S}$ such that these are consistent with respect to $\mathcal{I}$. Based on the Conjunctivity Theorem 13 we only approach the problem separately for the classes of constraints introduced in the previous subsection.

Moreover, we apply a specific kind of operational refinement in order to reduce the non-determinism of the resulting GCS. Whenever an arbitrary value of some proper value type is required, then we extend the input-type. However, this can not be applied to reduce the non-determinism arising from choice operations. In general, there exists a *choice normal form* for the GCS such that its components are the maximal deterministic operational refinements of the GCS that exist by Theorem 15.

In the following we restrict ourselves to the case of a single explicit constraint in addition to the one (trivial) uniqueness constraint that is required to assure value-representability and that has been used in [22] to construct *canonical update operations*. Then we look at an example with more than one constraint. We illustrate that although Theorem 13 holds, the structure of $S_{(\mathcal{I}_1 \wedge \mathcal{I}_2)}$ may be not at all obvious.

General (Path) Inclusion Constraints. Let $\mathcal{I}$ be a general inclusion constraint on C_1, C_2 defined via $c_i : T_{C_i} \rightarrow T$ $(i = 1, 2)$. Then each insertion into C_1 requires an additional insertion into C_2 whereas a deletion on C_2 requires a deletion on C_1. Update on one of the C_i requires an additional update on the other class.

Let us first concentrate on the insert-operation on C_1 (for an insert on C_2 there is nothing to do). Insertion into C_1 requires an input-value of type V_{C_1}; an additional insert on C_2 then requires an input-value of type V_{C_2}. However, these input-values are not independent, because the corresponding values of type T_{C_1} and T_{C_2} must satisfy the general inclusion constraint. Therefore we first show that the constraint can be "lifted" to a constraint on the value-representation types. Note that this is similar to the handling of IsA-constraints [22, Lemma 29].

Lemma 18. *Let C_1, C_2 be classes, $c_i : T_{C_i} \rightarrow T$ functions and let V_{C_i} be the value-representation type of C_i $(i = 1, 2)$. Then there exist functions $f_i : V_{C_i} \rightarrow T$ such that for all database instances $\mathcal{D}$*

$$f_1(d_1^{\mathcal{D}}(v_1)) = f_2(d_2^{\mathcal{D}}(v_2)) \;\Leftrightarrow\; c_1(v_1) = c_2(v_2) \tag{11}$$

for all $v_i \in codom_{\mathcal{D}}(x_{C_i})$ $(i = 1,2)$ holds. Here $d_i^{\mathcal{D}} : T_{C_i} \to V_{C_i}$ denotes the function used in the uniqueness constraint on C_i with respect to $\mathcal{D}$.

Proof. Due to Definition 7 we may define $f_i = c_i \circ (d_i^{\mathcal{D}})^{-1}$ on $c_i(codom_{\mathcal{D}}(x_{C_i}))$ $(i = 1,2)$.

Then we have to show that this definition is independent of the instance $\mathcal{D}$. Suppose $\mathcal{D}_1$, $\mathcal{D}_2$ are two different instances. Then there exists a permutation π on ID such that $d_i^{\mathcal{D}_2} = d_i^{\mathcal{D}_1} \circ \pi$, where π is extended to T_{C_i}. Then

$$c_i \circ (d_i^{\mathcal{D}_2})^{-1} = c_i \circ \pi^{-1} \circ (d_i^{\mathcal{D}_1})^{-1} = \pi^{-1} \circ c_i \circ (d_i^{\mathcal{D}_1})^{-1} \ ,$$

since c_i permutes with π^{-1}. Then the stated equality follows. □

Now let V_{C_1,C_2} be a subtype of $V_{C_1} \times V_{C_2}$ defined via the equality $f_1(v_1) = f_2(v_2)$, where $v_i :: V_{C_i}$ are the components of a value and f_i are the functions of Lemma 18. We omit the details. Then we can define the new insert-operation on C_1 by $(insert_{C_1})_{\mathcal{I}}(V :: V_{C_1,C_2}) ==$

$$insert_{C_1}(first(V)) \ ; \ insert_{C_2}(second(V)) \ . \tag{12}$$

Now we are able to generalize Theorem 8 with respect to general inclusion constraint.

Theorem 19. *Let $\mathcal{I}$ be a general inclusion constraint on C_1, C_2 defined via $c_i : T_{C_i} \to T$ and let $S_{\mathcal{I}}$ be the insert-operation of (12). Suppose that C_1 is not referenced by C_2. Then $S_{\mathcal{I}}$ is the GCS of the canonical insert-operation of Theorem 8 with respect to $\mathcal{I}$.*

Proof. We use the abbreviations $S_i = insert_{C_i}(V_i :: V_{C_i})$ $(i = 1,2)$. Then

$$wlp(S_{\mathcal{I}})(\mathcal{R}) \equiv \{V_1/first(V)\}.wlp(S_1)(\{V_2/second(V)\}.wlp(S_2)(\mathcal{R}) \ .$$

Since S_i is total and always terminating, we have $wlp(S_i) = wp(S_i)$. Since C_1 is not referenced by C_2, we know that S_2 is a $\{x_{C_2}\}$-operation. Therefore, $wlp(S_2)(\mathcal{R})$ is a logical combination of $\mathcal{R}$ without any substitution, hence $wlp(S_1)(\mathcal{R}) \Rightarrow wlp(S_{\mathcal{I}})(\mathcal{R})$. This proves (i) and (ii).

In particular $wlp(S_{\mathcal{I}})(\mathcal{I}) \equiv$

$$\{\, x_{C_1}/Union(x_{C_1}, Single(Pair(I_1, V_1))) \,,$$
$$\quad x_{C_2}/Union(x_{C_2}, Single(Pair(I_2, V_2))) \,\}.\mathcal{I}$$

with $I_1, I_2 :: ID$ and $V_i :: T_{C_i}$ with $c_1(V_1) = f_1(first(V))$ and $c_2(V_2) = f_2(first(V))$, where f_i are the functions of Lemma 18. Then property (iii) follows immediately. We omit the proof of (iv). □

Note there there is no need to require $C_1 \neq C_2$. Delete- and update-operations can be defined analogously to (12). Then a result analogous to Theorem 19 holds. We omit the details here. The generalization to path constraints is also straightforward.

(Path) Functional and Uniqueness Constraints. Now let $\mathcal{I}$ be a functional constraint on C defined via $c^1 : T_C \to T_1$ and $c^2 : T_C \to T_2$. In this case nothing is required for the delete operation whereas for inserts (and updates) we have to add a postcondition. Moreover, let $c^{\mathcal{D}} : T_C \to V_C$ denote the function associated with the value-representability of C and the database instance $\mathcal{D}$ and let all other notations be as before. Let us again concentrate on the insert-operation. Let $insert'_C$ denote the *quasi-canonical insert* on C [22]. Then we define

$$
\begin{array}{l}
(insert_C)_{\mathcal{I}}(V :: V_C) \;== \\
\quad I :: ID \;\mid\; I \leftarrow insert'_C(V)\;; \\
\quad V' :: T_C \;\mid\; member(Pair(I, V'), x_C) = true \;\to \\
\quad\; (\,\forall J :: ID, W :: T_C\,.\,(\,member(Pair(J, W), x_C) = true \\
\quad\;\; \wedge\, c^1(W) = c^1(V') \;\Rightarrow\; c^2(W) = c^2(V')\,) \;\to \\
\quad skip
\end{array}
\tag{13}
$$

Note that in this case there is no change of input-type.

Theorem 20. *Let $\mathcal{I}$ be a functional constraint on the class C defined via $c^1 : T_C \to T_1$ and $c^2 : T_C \to T_2$ and let $S_{\mathcal{I}}$ be the insert-operation of (13). Then $S_{\mathcal{I}}$ is the GCS of the canonical insert-operation on C defined by Theorem 8 with respect to $\mathcal{I}$.*

Proof. The proof is analogous to the one of Theorem 19. □

For delete- and update-operations an analogous result holds. We omit the details. The generalization to path constraints is also straightforward.

A uniqueness constraint defined via $c^1 : T_C \to T_1$ is equivalent to a functional constraint defined via c^1 and $c^2 = id : T_C \to T_C$ plus the trivial uniqueness constraint. Since trivial uniqueness constraints are already enforced by the canonical update operations, there is no need to handle separately arbitrary uniqueness constraints.

(Path) Exclusion Constraints. The handling of exclusion constraints is analogous to the handling of inclusion constraints. This means that an insert (update) on one class may cause a delete on the other, whereas delete-operations remain unchanged.

We concentrate on the insert-operation. Let $\mathcal{I}$ be an exclusion constraint on C_1 and C_2 defined via $c_i : T_{C_i} \to T$ $(i = 1, 2)$. Let $f_i : V_{C_i} \to T$ denote the functions from Lemma 18. Then we define a new insert-operation on C_1 by

$$
\begin{array}{l}
(insert_{C_1})_{\mathcal{I}}(V :: V_{C_1}) == \\
\quad insert_{C_1}(V)\;; \\
\quad \mu S.\,((I :: ID \;\mid\; V' :: T_{C_2} \;\mid\; member(Pair(I, V'), x_{C_2}) = true \\
\quad\;\; \wedge c^2(V') = f_1(V) \;\to\; delete_{C_2}(V')\;;\; S\,) \\
\quad \otimes\; skip\,)\;.
\end{array}
\tag{14}
$$

Theorem 21. *Let $\mathcal{I}$ be an exclusion constraint on the classes C_1 and C_2 defined via $c_i : T_{C_i} \to T$ $(i = 1, 2)$ and let $S_{\mathcal{I}}$ be the insert-operation of (14). Then $S_{\mathcal{I}}$ is the GCS of the canonical insert-operation on C_1 defined by Theorem 8 with respect to $\mathcal{I}$.*

Proof. The proof is analogous to the one of Theorem 19. □

For delete- and update-operations an analogous result holds. We omit the details. The generalization to path constraints is also straightforward.

(Path) Object Generating Constraints. Let $\mathcal{I}$ be an object generating constraint on a class C defined via the functions $c^i : T_C \rightarrow T_i$ $(i = 1, 2, 3)$. Then integrity enforcement requires to add additional inserts (deletes, updates) to each insert- (delete-, update-) operation. Let us illustrate the new insert-operation. The handling of delete and update-operations is analogous. As in the case of inclusion constraints we need a preliminary lemma.

Lemma 22. *Let $c^i : T_C \rightarrow T_i$ be functions $(i = 1, 2, 3)$ such that $c^1 \times c^2 \times c^3$ defines a uniqueness constraint on the class C. Then there exist functions $f_i : V_C \rightarrow T_i$ such that $c^i = f_i \circ c^{\mathcal{D}}$ holds for all instances $\mathcal{D}$, where $c^{\mathcal{D}} : T_C \rightarrow V_C$ corresponds to the value-representability of C.*

Proof. Since $c^1 \times c^2 \times c^3$ defines a uniqueness constraint on C, it follows from Definition 7 that there exists some $f : V_C \rightarrow T_1 \times T_2 \times T_3$ with $c^1 \times c^2 \times c^3 = f \circ c^{\mathcal{D}}$. Then $f_i = \pi_i \circ f$, where π_i is the projection to T_i $(i = 1, 2, 3)$, satisfy the required property. □

Then we define a new insert-operation on C as follows:

$$
\begin{array}{l}
(insert_C)_{\mathcal{I}}(V :: V_C) \;== \\
\quad \exists I' :: ID, V' :: T_C \,.\, member(Pair(I', V'), x_C) = true \\
\qquad\qquad \wedge\, c^1(V') = f_1(V) \;\rightarrow \\
\qquad (\,(\, V'' :: V_C \mid f_1(V'') = f_1(V) \wedge f_2(V'') = f_2(V) \\
\qquad\quad \wedge f_3(V'') = c^3(V') \;\rightarrow\; insert_C(V'')\,)\;; \\
\qquad\; (\, V''' :: V_C \mid f_1(V''') = f_1(V) \wedge f_2(V''') = c^2(V') \wedge \\
\qquad\quad f_3(V''') = f_3(V) \;\rightarrow\; insert_C(V''')\,)\,)\;; \\
\qquad\; \otimes\; skip\;)\;; \\
\\
\quad insert_C(V)
\end{array}
\tag{15}
$$

Theorem 23. *Let $\mathcal{I}$ be an object generating constraint on a class C defined via the functions $c^i : T_C \rightarrow T_i$ $(i = 1, 2, 3)$ such that $c^1 \times c^2 \times c^3$ defines a uniqueness constraint on the class C and let $S_{\mathcal{I}}$ be the insert-operation of (15). Then $S_{\mathcal{I}}$ is the GCS of the canonical insert-operation on C with respect to $\mathcal{I}$.*

Proof. The proof is analogous to the one of Theorem 19. □

For delete- and update-operations an analogous result holds. We omit the details. The generalization to path constraints is also straightforward.

Theorem 24. *Let $\mathcal{S}$ be a schema such that each class C in $\mathcal{S}$ is value representable. Suppose all explicit constraints in $\mathcal{S}$ are general inclusion constraints, exclusion constraints, functional constraints, uniqueness constraints, object generating constraints and path constraints. Then there exist generic update methods $insert_C$, $delete_C$ and $update_C$ for each class C in $\mathcal{S}$ that are consistent with respect to all implicit and explicit static constraints on $\mathcal{S}$.*

Proof. The result has been shown above in the Theorems 19–23 for a single explicit constraint. Then the general result follows from Theorem 13. □

5.3 Transforming Static Constraints into Transactions

Let us now consider the case of arbitrary methods which can be used to model transactions. We concentrate on the question whether it is possible to achieve general consistent methods as combinations of consistent primitive operations as e.g. given by Theorem 24. Regard the following simple example:

Example 3.

ACCOUNTCLASS ==
  `Structure` TRIPLE(PERSON,NAT,NAT)
`End` ACCOUNTCLASS

`Constraints`
  $Pair(I,V) \in \text{ACCOUNTCLASS} \Rightarrow second(V) + third(V) \geq 0$
  $Pair(I,V) \in \text{ACCOUNTCLASS} \wedge Pair(I',V') \in \text{ACCOUNTCLASS} \wedge$
    $first(V) = first(V') \Rightarrow I = I'$

The second component gives the account of a person and the third component gives his/her credit limit. Let

$$transfer(P_1 :: PERSON, P_2 :: PERSON, T :: NAT) ==$$
$$I_1, I_2 :: ID, N_1, N_2, M_1, M_2 :: NAT \mid$$
$$Pair(I_1, Triple(P_1, N_1, M_1)) \in \text{ACCOUNTCLASS} \wedge$$
$$Pair(I_2, Triple(P_2, N_2, M_2)) \in \text{ACCOUNTCLASS} \rightarrow$$
$$update_{AccountClass}(P_1, Triple(P_1, N_1 - T, M_1)) ;$$
$$update_{AccountClass}(P_2, Triple(P_2, N_2 + T, M_2))$$

In this case the precondition to be added is simply the weakest precondition, i.e. $N_1 + M_1 - T \geq 0$. Now regard the operation

$$S = transfer(P_1, P_2, T_1) ; transfer(P_2, P_1, T_2) \ .$$

The precondition to be added to S in order to enforce the first constraint simply is $N_1 + M_1 - T_1 + T_2 \geq 0 \wedge N_2 + M_2 - T_2 + T_1 \geq 0$, which is trivially satisfied if $T_1 = T_2$. However, for large values of $T_1 = T_2$ this condition is weaker then adding three precondition separately. □

Theorem 25. *Let S be a method and let $S_1 \ldots S_n$ be canonical update operations on a schema $\mathcal{S}$ occurring within S. Let S' result from S by replacing each S_i by its GCS $(S_i)_{\mathcal{I}}$ with respect to some static constraint $\mathcal{I}$. Then $S_{\mathcal{I}}$ and S' are in general not semantically equivalent.*

Proof. A counterexample has been given in Example 3. □

As a consequence of this theorem it is even not sufficient to know explicitly the GCS of basic update operations with respect to a single constraint. Although Theorem 13 allows the GCS with respect to more than one constraint to be built successively, we have seen in Theorems 19–23 that the GCS with respect to one constraint is no longer a basic update operation. Let us illustrate this open problem by a simple example.

Example 4. Let C_1, C_2, C_3 be classes and $c_i : T_{C_i} \rightarrow T$ $(i = 1, 2, 3)$ be functions and suppose there are defined

- a general inclusion constraint $\mathcal{I}_1$ on C_1 and C_2 via c_1 and c_2,
- a general inclusion constraint $\mathcal{I}_2$ on C_1 and C_3 via c_1 and c_3 and
- an exclusion constraint $\mathcal{I}_3$ on C_2 and C_3 via c_2 and c_3.

Clearly, the GCS of the insert-operation on C_1 with respect to $\mathcal{I}_1 \wedge \mathcal{I}_2 \wedge \mathcal{I}_3$ is $fail$, which is hard to build successively. □

6 Enforcing Transition Integrity

In Section 4 we discussed integrity enforcement in a general framework with respect to static constraints. Let us now try to generalize the results for transition constraints. Thus, let S be an X-operation and $\mathcal{J}$ a transition constraint on Y with $X \subseteq Y$. The idea is again to construct a "new" Y-operation $S_{\mathcal{J}}$ that is consistent with respect to $\mathcal{J}$ and specializes S. Again this leads to the idea to construct a *Greatest Consistent Specialization* of S with respect to $\mathcal{J}$. The difference to the case of static constraints is the use of a different proof obligation for consistency according to Definition 4.

6.1 GCSs with Respect to Transition Constraints

We start giving a formal definition of a *Greatest Consistent Specialization* (GCS) of an operation S with respect to a transition constraint $\mathcal{J}$.

Definition 26. Let $X \subseteq Y$ be state spaces, S an X-operation and $\mathcal{J}$ a transition integrity constraint on Y. A Y-operation $S_{\mathcal{J}}$ is a *Greatest Consistent Specialization* (GCS) of S with respect to $\mathcal{J}$ iff

(i) $wlp(S)(\mathcal{R}) \Rightarrow wlp(S_{\mathcal{J}})(\mathcal{R})$ holds on Y for all formulae $\mathcal{R}$ with $fr(\mathcal{R}) \subseteq X$,
(ii) $wp(S)(true) \Rightarrow wp(S_{\mathcal{J}})(true)$ holds on Y,
(iii) $wlp(\Phi(\mathcal{J}))(\mathcal{R}) \Rightarrow wlp(S_{\mathcal{J}})(\mathcal{R})$ holds on Y for all formulae $\mathcal{R}$ with $fr(\mathcal{R}) \subseteq Y$ and
(iv) for each Y-operation T satisfying properties (i) – (iv) (instead of $S_{\mathcal{J}}$) we have
  (a) $wlp(S_{\mathcal{J}})(\mathcal{R}) \Rightarrow wlp(T)(\mathcal{R})$ for all formulae $\mathcal{R}$ with $fr(\mathcal{R}) \subseteq X$ and
  (b) $wp(S_{\mathcal{J}})(true) \Rightarrow wp(T)(true)$.

Note that properties (i), (ii) and (iii) say that $S_{\mathcal{J}} \sqsubseteq S$, where $\sqsubseteq$ is the specialization order of Definition 4. Property (iv) requires $S_{\mathcal{J}}$ to be consistent with respect to the constraint $\mathcal{J}$. Finally, property (v) states that each consistent specialization T of S with $T \sqsubseteq S$ also specializes $S_{\mathcal{J}}$.

Based on this formal definition of a GCS we can now raise the same questions as in the static case. Before we state the result on the (unique) existence of GCSs, let us first examine the relation between static and transition constraints. Suppose $\mathcal{I}$ is a static constraint on Y, then we may regard $\mathcal{I}$ also as a transition constraint. Let $\mathcal{I}'$ result from $\mathcal{I}$ by replacing each state variable x_i occurring freely in $\mathcal{I}$ by x_i'. Then $\mathcal{I} \Rightarrow \mathcal{I}'$ defines the *corresponding transition constraint* denoted as $\mathcal{J}_{\mathcal{I}}$. Then we know that consistency with respect to $\mathcal{I}$ is equivalent to consistency with respect to $\mathcal{J}_{\mathcal{I}}$. This implies the following result:

Theorem 27. *Let S be an X-operation and $\mathcal{I}$ a static constraint on Y with $X \subseteq Y$. If $S_{\mathcal{I}}$ and $S_{\mathcal{J}_{\mathcal{I}}}$ are the GCSs of S with respect to $\mathcal{I}$ and the transition constraint $\mathcal{J}_{\mathcal{I}}$ respectively, then these two GCSs are semantically equivalent.*

Proof. This follows directly from Definitions 9 and 26. Using the equivalence of consistency proof obligations with respect to $\mathcal{I}$ and $\mathcal{J}_{\mathcal{I}}$ implies the two definitions to coincide. □

Next, we are able to proof the (unique) existence of a GCS also for transition constraints. As in the static case the proof will be non-constructive, hence does not help to construct a GCS.

Theorem 28. *Let S be an X-operation, $X \subseteq Y$ and $\mathcal{J}$ a transition integrity constraint on Y. Then there exists a greatest consistent specialization $S_{\mathcal{J}}$ of S with respect to $\mathcal{J}$. Moreover, $S_{\mathcal{J}}$ is uniquely determined (up to semantic equivalence) by S and $\mathcal{J}$.*

Proof. The proof is analogous to the one of Theorem 11. □

6.2 Compatibility Results

Let us now address the compatibility problems with respect to the conjunction of constraints, inheritance and refinement. Note that due to Theorem 27 some of the results in Section 4.2 occur as special cases of the results here.

Theorem 29. *Let $\mathcal{J}_1$ and $\mathcal{J}_2$ be transition constraints on Y_1 and Y_2 respectively. If for any operation S the GCS with respect to $\mathcal{J}_i$ is denoted by $S_{\mathcal{J}_i}$ $(i = 1, 2)$, then for any X-command S with $X \subseteq Y_1 \cap Y_2$ the GCSs $(S_{\mathcal{J}_1})_{\mathcal{J}_2}$ and $(S_{\mathcal{J}_2})_{\mathcal{J}_1}$ are semantically equivalent.*

Proof. The proof is analogous to Theorem 12. □

Theorem 30. *Let $S_{\mathcal{J}}$ be the GCS of the X-operation S with respect to the transition constraint $\mathcal{J}$ defined on Y with $X \subseteq Y$. Let T be a Z-operation that specializes S. If $\mathcal{J}$ is regarded as a constraint on $Y \cup Z$, then the GCS $T_{\mathcal{J}}$ of T with respect to $\mathcal{J}$ is a specialization of $S_{\mathcal{J}}$.*

Proof. The proof is analogous to the one of Theorem 14. □

Theorem 31. *The GCS $S_{\mathcal{J}}$ of a deterministic X-operation S with respect to a transition constraint $\mathcal{J}$ is in general non-deterministic.*

Proof. This follows from Theorem 27, since determinism is not even preserved by GCSs with respect to static constraints as shown by the counter-example in Example 2. □

Hence the problem remains to remove the non-determinism. Due to [20, Proposition 4.2] it is easy to see that whenever S is consistent with respect to a transition constraint $\mathcal{J}$, then each specialization T of S does so, too.

Theorem 32. *Let $S_{\mathcal{J}}$ be the GCS of a deterministic operation S with respect to the transition constraint $\mathcal{J}$ and let T be some deterministic operational refinement of $S_{\mathcal{J}}$. Then T specializes S. Moreover, if T' is any deterministic specialization of S that is consistent with respect to $\mathcal{J}$, then T' also specializes some deterministic operational refinement T of $S_{\mathcal{J}}$.*

Proof. The proof is analogous to Theorem 15. □

7 Conclusion

In this paper we addressed the problem of integrity enforcement in object-oriented databases. First we introduced a formally defined object-oriented data-model (OODM) that includes inclusion and referential constraints implicitly in a schema and allows in particular explicit static and transition constraints to be defined. We gave formally defined general proof obligations for static and

transition consistency. Using a general purpose theorem prover to prove these formulae gives a verification based approach to consistency.

However, the verification approach does not help in writing consistent methods. Hence the idea to replace each update operation (methods in the OODM) by a *Greatest Consistent Specialization* (GCS). We show that such GCSs always exist and are uniquely determined by the operation and the constraints up to semantic equivalence. Moreover, they are compatible with the conjunction of constraints, inheritance and refinement.

An unsolved open problem is the general structure of a GCS including the problem of its computation. One idea is to exploit the structure of a constraint that must involve constructor- and selector-expressions on state vaiables (classes in the OODM). However, this idea is still two vague to be discussed in this paper. Another open problem concerns the generalization to arbitrary dynamic constraints. Lipeck has shown in [16] that dynamic constraints expressed in some generalized propositional temporal logic give rise to transition graphs. It should be possible to derive a suitable proof obligation in the predicate transformer calculus also for such constraints. Then the idea of GCS construction should carry over even to this class of constraints that comprises static and transition constraints.

For the case of the OODM we achieve more specific results for a wide class of static constraints that occur as generalizations of kinds of constraints known from the relational model. We show that integrity enforcement for primitive update operations is possible. Moreover, part of the non-determinism can be removed in a canonical way by extension of the required input-type. In general, however, this is not sufficient, since it is not possible to construct the GCS as a combination of the GCSs of primitive operations.

References

1. S. Abiteboul: *Towards a deductive object-oriented database language*, Data & Knowledge Engineering, vol. 5, 1990, pp. 263 – 287
2. S. Abiteboul, P. Kanellakis: *Object Identity as a Query Language Primitive*, in Proc. SIGMOD, Portland Oregon, 1989, pp. 159 – 173
3. A. Albano, A. Dearle, G. Ghelli, C. Marlin, R. Morrison, R. Orsini, D. Stemple: *A Framework for Comparing Type Systems for Database Programming Languages*, in Type Systems and Database Programming Languages, University of St. Andrews, Dept. of Mathematical and Computational Sciences, Research Report CS/90/3, 1990
4. M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik: *The Object-Oriented Database System Manifesto*, Proc. 1st DOOD, Kyoto 1989
5. S. Al Fedaghi, B. Thalheim: *Fundamentals of databases - The key concept*, submitted for publication, 1990
6. C. Beeri: *Formal Models for Object-Oriented Databases*, Proc. 1st DOOD 1989, pp. 370 – 395
7. C. Beeri: *A formal approach to object-oriented databases*, Data and Knowledge Engineering, vol. 5 (4), 1990, pp. 353 – 382

8. C. Beeri, Y. Kornatzky: *Algebraic Optimization of Object-Oriented Query Languages*, in S. Abiteboul, P. C. Kanellakis (Eds.): Proc. ICDT '90, Springer LNCS 470, pp. 72 – 88

9. E. W. Dijkstra, C. S. Scholten: *Predicate Calculus and Program Semantics*, Springer-Verlag, 1989

10. H.-D. Ehrich, M. Gogolla, U. Lipeck: *Algebraische Spezifikation abstrakter Datentypen*, Teubner-Verlag, 1989

11. H.-D. Ehrich, A. Sernadas: *Fundamental Object Concepts and Constructors*, in G. Saake, A. Sernadas (Eds.): Information Systems – Correctness and Reusability, TU Braunschweig, Informatik Berichte 91-03, 1991

12. H. Ehrig, B. Mahr: *Fundamentals of Algebraic Specification*, vol.1, Springer 1985

13. P. Fraternali, S. Paraboschi, L. Tanca: *Automatic Rule Generation for Constraint Enforcement in Active Databases*, in U. Lipeck, B. Thalheim (Eds.): Proc. 4th Int. Workshop on Foundations of Models and Languages for Data and Objects "MODELLING DATABASE DYNAMICS", Volkse (Germany), October 19-22, 1992, in this issue

14. A. Heuer: *Objektorientierte Datenbanksysteme*, Addison Wesley, 1992

15. S. Khoshafian, G. Copeland: *Object Identity*, Proc. 1st Int. Conf. on OOPSLA, Portland, Oregon, 1986

16. U. W. Lipeck: *Dynamische Integrität von Datenbanken* (in German), Springer IFB 209, 1987

17. G. Nelson: *A Generalization of Dijkstra's Calculus*, ACM TOPLAS, vol. 11 (4), October 1989, pp. 517 – 561

18. G. Saake, R. Jungclaus: *Specification of Database Applications in the TROLL Language*, in D. Harper, M. Norrie (Eds.): Proc. Int. Workshop on the Specification of Database Systems, Glasgow, 1991

19. K.-D. Schewe, J. W. Schmidt, I. Wetzel, N. Bidoit, D. Castelli, C. Meghini: *Abstract Machines Revisited*, FIDE Technical Report 1991/11, February 1991

20. K.-D. Schewe, I. Wetzel, J. W. Schmidt: *Towards a Structured Specification Language for Database Applications*, in D. Harper, M. Norrie (Eds.): Proc. Int. Workshop on the Specification of Database Systems, Springer WICS, 1991, pp. 255 – 274

21. K.-D. Schewe, B. Thalheim, I. Wetzel, J. W. Schmidt: *Extensible Safe Object-Oriented Design of Database Applications*, University of Rostock, Report CS - 09 - 91, September 1991

22. K.-D. Schewe, J. W. Schmidt, I. Wetzel: *Identification, Genericity and Consistency in Object-Oriented Databases*, in J. Biskup, R. Hull (Eds.): Proc. ICDT '92, Springer LNCS 646, 1992, pp. 341 – 356

23. K.-D. Schewe: *Class Semantics in Object Oriented Databases*, submitted 1992

24. B. Thalheim: *Dependencies in Relational Databases*, Teubner Leipzig, 1991

25. J. Van den Bussche, Dirk Van Gucht: *A Hierarchy of Faithful Set Creation in Pure OODBs*, in J. Biskup, R. Hull (Eds.): Proc. ICDT '92, Springer LNCS 646, 1992, pp. 326 – 340

This article was processed using the LaTeX macro package with LLNCS style