INSTITUT FÜR INFORMATIK

UND PRAKTISCHE MATHEMATIK

Functional Dependencies for Object Databases: Motivation and Axiomatization

Hans-Joachim Klein, Jochen Rasch

Bericht Nr. 9706

Oktober 1997

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Institut für Informatik und Praktische Mathematik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Functional Dependencies for Object Databases: Motivation and Axiomatization

Hans-Joachim Klein, Jochen Rasch

Bericht Nr. 9706
Oktober 1997

e-mail: hjk@informatik.uni-kiel.de
jor@informatik.uni-kiel.de

Dieser Bericht ist als persönliche Mitteilung aufzufassen.

Abstract

Object identification by abstract identifiers should be considered as a modeling and not as a database concept. This means that object identifiers are not appropriate for the access to specific objects using a database language. In this paper we discuss how the relational concept of a functional dependency can be adapted to object databases in order to get more convenient ways of accessing objects. Graph based object functional dependencies are proposed as a means to specify constraints between attributes and object types of an object schema. Value based identification criteria can be defined using a special type of object functional dependencies. Different definitions of satisfaction are given for these constraints, based on a so-called validation relation, and their relationships are investigated. These definitions are related to different forms of identification. Using the strongest notion of satisfaction, inference rules for the derivation of new dependencies are discussed with emphasis on the characteristics of rules combining two dependencies, like the transitivity rule. In addition to generalized relational rules further rules are needed, mainly concerned with transition from the object type level to the attribute level and vice versa.

Keywords: object oriented data models, object functional dependencies, value based identification, identification criteria, keys, inference rules

Contents

Functional Dependencies for Object Databases: Motivation and Axiomatization

1 Introduction

One of the fundamental concepts of object orientation is object identity ([CK86]). It allows to distinguish objects even if they coincide in their values. In data models this abstract concept is often realized by object identifiers or surrogates, i. e. by internal identifiers (e. g. [AH84], [AK89], [Cod79], [HOT76], [HY91], [MW90]). Here, internal means that the query language of a data model does not provide direct access to the identifiers. Sometimes query languages have the capability of retrieving an arbitrarily chosen object from a set of objects, comparable to selecting a copy of a book to be borrowed from a number of copies available in a library. This, however, is a random selection and no identifying access. In general, applying the retrieval operation twice to the same set would result in two different objects.

A major goal of database design is to define object and relationship types in such a way that they represent meaningful units of information with respect to the semantics of the underlying application domain. Thus, it should be possible to address objects by specifying some of their properties at the level of values and relationships according to these semantic units. This shows that object identifiers are a modeling concept and not a database concept ([WdJ91]). For this reason starting the search for an object with a set of values should be possible. Identifying values either give direct access to a single object or they define objects as starting points for the retrieval of an object by navigation along links between objects in the given database.
Especially in the case of object oriented database design and relational implementation, knowledge about the accessibility of objects based on their data values is of great interest. Application domains often suggest natural value based identification criteria (VBICs). From the user's point of view these criteria are preferable to abstract object identifiers since they carry semantics of the modeled domain.
The significance of value based identification mechanisms for objects has been emphasized by several authors (e.g. [AVdB95], [Bee93], [Gog95], [Kim95], [ST93]). However, less work has been done in characterizing and investigating reasonable forms of VBICs, together with their interaction. A further interesting problem is how the structuring of data in an object oriented schema, e. g. inheritance, influences value based identification.

Some examples of VBICs to be covered by a more general theory of constraints on both object and value level are given in Figures 1 and 2: hotels offer rooms in different categories, e. g. single and double rooms (Figure 1 (a)). The values of *Accommodation* attributes cannot be used as entry values to access a single *Accommodation* object because different hotels may offer rooms in the same category. A *Hotel* object together with a value for *room_category*, however, uniquely identifies an *Accommodation* object. If *name* is a key for *Hotel*, i. e. *Hotel* objects are identified by their *name* value, *room_category* and *name* can serve as a VBIC for *Accommodation*. Consider the schema in Figure 1 (b), representing information about different branches of

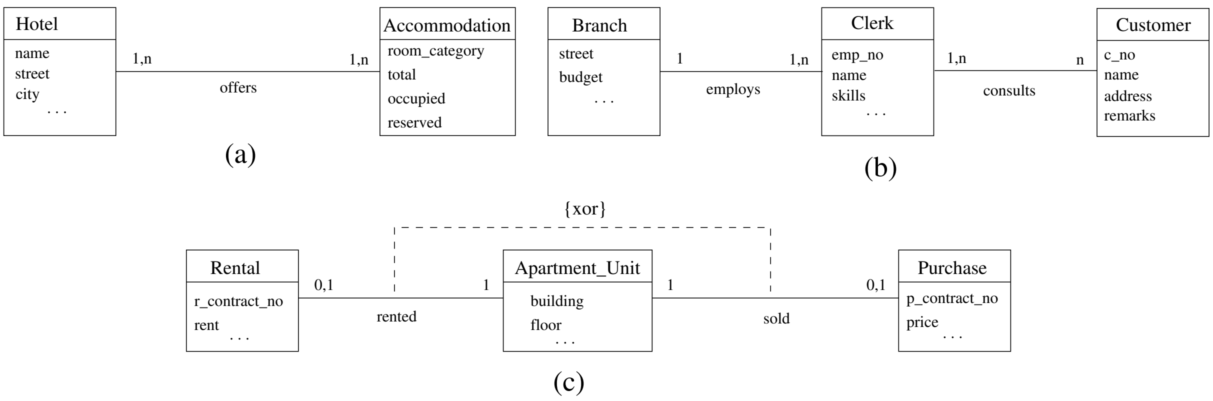


Figure 1: Some examples of VBICs

a bank. A customer may have accounts at different branches. At each branch one clerk is assigned to her or him as investment consultant. Therefore *Branch* and *Customer* objects together determine *Clerk* objects. If *street* and *c_no* are keys for *Branch* and *Customer*, respectively, a VBIC for *Clerk* is given. This provides an additional identification criterion besides the obvious identification of *Clerk* by *emp_no*, which may be useful for some applications. Neither *street* nor *c_no* alone is sufficient to identify *Clerk* objects.

Figure 1 (c) shows an example of a different form of identification: apartments in a building are either rented or condominium apartments. Therefore the number of the lease or the sales contract can be used to distinguish *Apartment_Unit* objects by value, leading to a "disjunctive identification" of apartments either by *Rental* or by *Purchase*. The exclusive-or constraint between the relationships *rented* and *sold* is expressed by the dotted line with constraint type specified as $\{xor\}$ (cf. [Rat97]).

Assume that a building contractor runs, via subcompanies, some of the apartments he builds. He offers this service also to private investors who bought an apartment but do not occupy it themselves. So a *Rental Institution* is specialized either to *Investor* or to *Administering Company*, as shown by the mandatory inheritance hierarchy in Figure 2. *Investor* is a specialization of type *Buyer* and *Administering Company* an optional specialization of type *Subcompany* (not every subcompany is an owner of apartments). *Buyer* and *Subcompany* objects are distinguishable by their *buyer_no* and *company_no* value, respectively. The attributes are inherited to the specialized types therefore also providing VBICs for *Investor* and *Administering Company*, respectively. In this scenario, value based identification of *Rental Institution* objects becomes

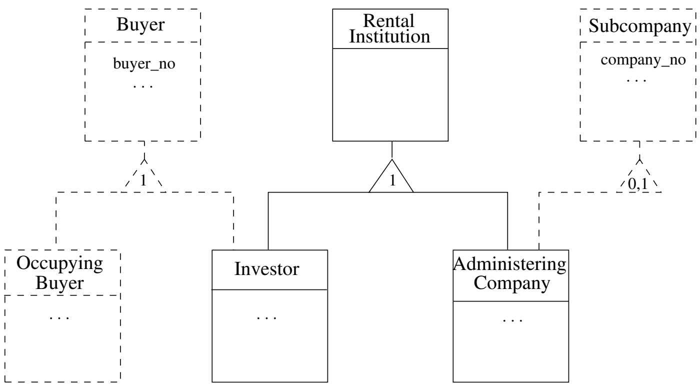


Figure 2: VBIC through inheritance

possible since every *Rental Institution* object is specialized. This leads to "identification by generalization/specialization", similar to the disjunctive criteria presented in Figure 1 (c).

One simple approach to provide value based identification of objects (especially concerning relational implementation) is to introduce an artificial identifier attribute ([HY91], [RBP$^+$91]), i. e. to make the abstract identifier visible. This approach should not be regarded as value based identification in the original sense because it does not refer to the values and relationships of the objects themselves. The other extreme is to use the complete object value for identification, including references to other objects recursively ([AK89], [AVdB95], [DV93]), leading to the notion of deep equality of objects. This is a very general way to identify objects by values. However, deep equality takes into account only references starting from an object and therefore does not consider the complete "relationship environment" of an object, since references directed towards the object are ignored. From a practical point of view the use of the complete object value as a VBIC, including all relationships to other objects, is inappropriate. Thus, the question how to determine VBICs similar to the key concept of the relational data model is raised. A reasonable solution should lie between these two extremes by exploiting all features of the object oriented data model.
In [Che76] a relationship with cardinality restricted to 1, i. e. representing a function between entity sets, may contribute to obtain a key for an entity type (so-called *weak entity type*, similar to the example in Figure 1 (a)) by using the key of another entity type related to it. The idea of using relationships to find VBICs was applied in [Zan79] to determine keys for records in a network schema. There not only record types of a single set type were taken into account but also record types reachable via a sequence of set types. In [ST93] this proceeding was applied to object oriented schemas using a sequence of relationships between classes, including inheritance. More general approaches to determine keys are presented by observation formulas and object terms in [AVdB95] and [Gog95], respectively. However, there is no further consideration on how to determine the proposed terms and how to distinguish different kinds of identification terms. In the following we propose an approach to this matter by generalizing functional dependencies to object schemas (cf. [KR97]).

The paper is organized as follows: the next section introduces some basic notions of the object model we use, including a formalization of the terms object schema and schema graph, as well as some concepts of the relational data model. Sections 3 and 4 describe our approach to generalize functional dependencies to object functional dependencies. These graph based constraints are spanning trees of subgraphs of the schema graph. Different semantics for object functional dependencies, corresponding to different notions of identification, are discussed in Section 5, including some remarks on effects that have to be taken into account if dependencies involving more than one object type are considered. In Section 6 inference rules for object functional dependencies generalizing the well-known relational rules are investigated together with additional inference rules which become necessary for object functional dependencies. We conclude with some remarks on related topics and an outlook on future work.

2 Preliminaries

The object model used in our approach is similar to the static part of the Object Modeling Technique ([RBP$^+$91], [Rat97]). We start with a formal description of the model.

2.1 Object schemas

An object schema consists of a finite number of named object types and binary relationships between these types, including inheritance. For our purposes it is sufficient to consider an inheritance hierarchy as a set of binary relationships with additional cardinality constraints for mandatory and optional hierarchies. An object type is a pair $(O, attr(O))$, where O is the type name and $attr(O)$ is a finite set of attributes with a non empty atomic domain assigned to each attribute. In the following we will identify an object type with its name in order to simplify notation. Relationships may have cardinalities as additional constraints. For simplicity we assume the names of object types, relationships, and attributes to be unique throughout the schema.

Let I be a countable infinite set of object identifiers. An object o of type O is a pair $o = (i, v)$ with $i \in I$ and v a tuple over $attr(O)$, called object value. The extension $ext(O)$ of an object type O is a finite set of objects of type O. Identifiers are assumed to be unique within $ext(O)$. Let $I(ext(O))$ denote the set of object identifiers occurring in $ext(O)$. The extension $ext(r)$ of a relationship r between two object types O_1, O_2 with extensions $ext(O_1)$, $ext(O_2)$ is a finite set of links $(i_1, i_2) \in I \times I$ where $i_1 \in I(ext(O_1))$, $i_2 \in I(ext(O_2))$. If a cardinality constraint is specified for r then each extension of r has to comply with the constraint. The state $s(\mathcal{S})$ of an object schema $\mathcal{S}$ consists of an extension for every object type and every relationship of $\mathcal{S}$ where $I(ext(O_1)) \cap I(ext(O_2)) = \varnothing$ for each pair of different object types O_1, O_2 of $\mathcal{S}$. The extension of an object type O can be represented by a relation over attribute set $\{id_O\} \cup attr(O)$, where id_O is an attribute with domain I. With this view the abstract object identifier becomes visible.

The schema graph $G_\mathcal{S}$ of an object schema $\mathcal{S}$ is a labeled graph (V, E, l) where the set of nodes V corresponds to the object types of $\mathcal{S}$ and where the set of edges E represents the relationships of $\mathcal{S}$: $\{O_1, O_2\} \in E$ iff a relationship between O_1 and O_2 exists in $\mathcal{S}$. l is an edge-labeling function with $l(e) = r$ iff $e \in E$ corresponds to relationship r of $\mathcal{S}$. A path in $G_\mathcal{S}$ is a sequence $\Pi = O_1 e_1 O_2 \ldots e_{n-1} O_n$ with $O_i \in V$, $e_j \in E$, $e_j = \{O_j, O_{j+1}\}$, $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, n-1\}$. Let a state for schema $\mathcal{S}$ be given. A link chain (with respect to Π) between objects $o_1 \in ext(O_1)$ and $o_n \in ext(O_n)$ is a sequence $\pi = o_1 l_1 o_2 \ldots l_{n-1} o_n$ such that $o_i \in ext(O_i)$ and $l_j \in ext(r_j)$, where l_j is the pair of identifiers of o_j and o_{j+1}, and r_j is the label of edge e_j. For an object type O, let $sets(O) := 2^{attr(O)} \cup \{\{O\}\}$ and $sets^+(O) := sets(O) \setminus \{\varnothing\}$. O will be written as shorthand notation for the singleton $\{O\}$. For a given object schema $\mathcal{S}$, let $OT_\mathcal{S}$ be the set of all object types of $\mathcal{S}$ and $\mathcal{D}_\mathcal{S} := \bigcup_{O \in OT_\mathcal{S}} sets(O)$. We shall assume an object schema to be non empty, i. e. it contains at least an object type.

2.2 Relational notions

Let $\beta = \{B_1, \ldots, B_m\}$, $m \geq 1$, be a finite set of attributes with domain function $dom: \beta \to \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$, $k \geq 1$, where each $\mathcal{D}_i$ is a non empty domain of atomic values. A tuple over β is a function $t: \beta \to \bigcup_{i=1}^{k} \mathcal{D}_i$ with $t(B) \in dom(B)$ for each $B \in \beta$. In examples it will be written in the usual tuple notation. A partial tuple t over β is a tuple over β with $t(B) \in dom(B)$ or $t(B)$ undefined. Undefined values are represented by the special symbol $'-'$. A (partial) relation over β is a finite set of (partial) tuples over β. Tuple t over β is total on $\zeta \subseteq \beta$ iff t is defined for each $C \in \zeta$. t is undefined on ζ iff t is undefined for each attribute $C \in \zeta$. $t|_\zeta$ denotes the restriction of t to attribute set ζ. For a relation R over β and an attribute set $\beta' \subseteq \beta$, the strong null filter $SNF(R, \beta')$

denotes the set of all tuples of R which are total on β'. The weak null filter $WNF(R,\beta')$ denotes the set of all tuples of R which are not undefined on β'. For a relation R, α_R denotes the set of attributes of R. If R is a relation and $\beta \subseteq \alpha_R$, then $R[\beta]$ denotes the usual projection of R onto β. Let t, t' be partial tuples over β. t' subsumes t on $\zeta \subseteq \beta$ iff $(\forall C \in \zeta)(t'(C) = t(C) \vee t(C) = '-')$. To give an example, let $t_1 = (1,2,3)$, $t_2 = (1,2,-)$, $t_3 = (1,-,3)$, and $t_4 = (-,2,-)$ be tuples over the same attribute set. Then t_1 subsumes t_1, t_2, t_3, and t_4, t_2 subsumes t_2 and t_4, but neither does t_2 subsume t_3 nor does t_3 subsume t_2.

Let $ext(O)$ be an extension of an object type O. The relational representation $rel_{ext(O)}$ of $ext(O)$ is the relation R with $\alpha_R = \{id_O\} \cup attr(O)$ and $R = \{t | t \text{ tuple over } \alpha_R \wedge (\exists (i,v) \in ext(O))(t(id_O) = i \wedge t|_{attr(O)} = v)\}$. id_O is called identifier attribute. The attributes from $attr(O)$ are called value attributes. The domains of the relational attributes are given by the domains of the corresponding object type attributes. For every relationship r between object types O_1 and O_2, $ext(r)$ is a set of links which may be considered as relational representation of itself with attribute set $\{id_{O_1}, id_{O_2}\}$, denoted as $rel_{ext(r)}$.

3 Extending functional dependencies to objects

A straightforward approach to the specification of VBICs for objects types is the generalization of functional dependencies (FDs) known from the relational model. This proceeding allows to take advantage of the well-founded theory of FDs.

Consider a relation R with attribute set α_R and an FD $f: \beta \rightarrow \gamma$ with $\beta, \gamma \subseteq \alpha_R$. f refers only to attributes of α_R and whether f is satisfied in a given relational database state is dependent only on R and independent of any further relations in the state. A straightforward application of this concept to object schemas leads to a constraint on object type level, i.e. left and right side of an FD f' may not only refer to attributes of O but also to the object type itself, by regarding the internal identifier as a special attribute: $f' : \beta' \rightarrow \gamma'$ with β', $\gamma' \subseteq attr(O) \cup \{O\}$ for some object type O of a schema $\mathcal{S}$. This allows to express constraints stating that objects of O are distinguishable (identifiable) by their value or a part thereof in the same way as tuples in a relation can be distinguished by looking at their values in key attributes. f' is restricted to local identification of O by its own attributes only, like common FDs as intra-relational integrity constraints on a single relation type. However, if the attribute values of an object are not sufficient to identify it in the extension of O in a state $s(\mathcal{S})$, relationships to other objects and the values of these objects in $s(\mathcal{S})$ can be taken into account. The simplest example for this kind of identification is the weak entity concept of the Entity-Relationship Model ([Che76]). Generalizing this approach leads to FDs of the form $f'' : \Delta \rightarrow \Gamma$, with $\Delta, \Gamma \subseteq \mathcal{D}_{\mathcal{S}}$, where any object type of $\mathcal{S}$ may contribute to Δ and Γ, resulting in a dependency on schema level similar to inter-relational constraints. Object types between which such kind of dependency exists do not have to be directly connected in the schema graph $G_{\mathcal{S}}$, a path between them is sufficient.

Between any two object types appearing in Δ or between any object type of Δ and a type appearing in Γ more than one path may exist in $G_{\mathcal{S}}$. In schema (a) from Figure 3 for example, there are two different paths (via p, q and via r, s, t) for the FD $\{O_1\} \rightarrow \{O_3\}$, connecting O_1 and O_3. For schema (b) and FD $\{O_1, D\} \rightarrow \{L\}$, two paths connecting O_3 with O_6 and two paths between O_1 and O_3 can be found. Usually, different paths correspond to different semantics and a database modeler has one of these paths in mind when specifying a dependency (cf. [Lie82]). This ambiguity is also known from the universal relation approach ([MUV84]). Obviously an

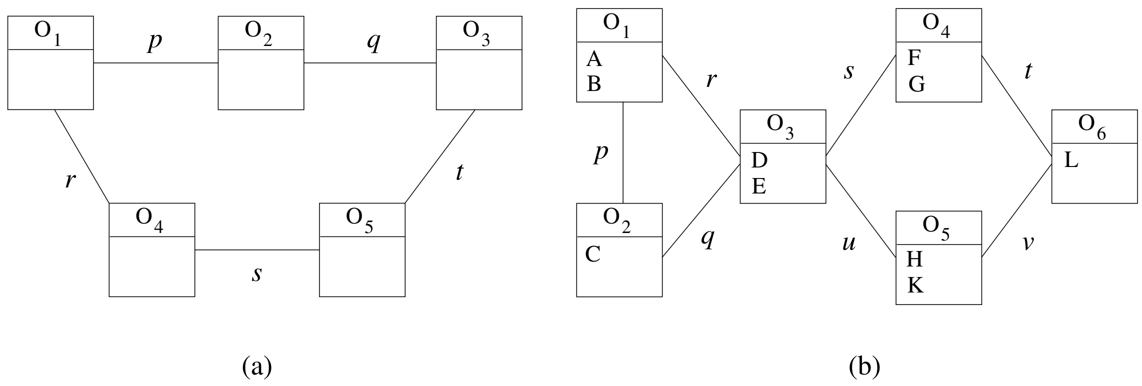


Figure 3: Ambiguity of an FD at schema level

FD can be satisfied with respect to one path and not with respect to another one, as shown by the following example:

Example 3.1: Consider the object types O_1, O_2, O_3 from Figure 3 (b) with extensions

$$\begin{array}{ll} ext(O_1) = \{(1,v_1),(2,v_2),(3,v_3)\}, & ext(O_2) = \{(4,v_4),(5,v_5),(6,v_6)\}, \\ ext(O_3) = \{(7,v_7),(8,v_8)\}, & ext(p) = \{(1,4),(2,5),(3,6)\}, \\ ext(r) = \{(1,7),(2,7),(3,8)\}, & ext(q) = \{(4,7),(5,8),(6,8)\}. \end{array}$$

An FD $g : \{O_1\} \to \{O_2\}$ via relationship p obviously holds for the given extensions since $ext(p)$ represents a one-to-one relationship. Using relationships r and q, g does not hold because objects $(5,v_5)$ and $(6,v_6)$ have links to the same O_3- and O_1-object and from there cannot be distinguished by looking at the O_1-objects associated to them. □

On the other hand there has to exist at least one path between each pair of object types appearing in Δ and also a path between the types of Γ and the types of Δ. Otherwise, f'' would state a dependency on object types without a path of relationships between these, i. e. between types in different components of $G_{\mathcal{S}}$. Thus, paths have to be specified together with FDs.

For a generalization of FDs, a graph based approach seems to be appropriate to deal with the mentioned problems, leading to the notion of an *object functional dependency*. Here we concentrate on a restricted form of such dependencies without cycles.

Definition 3.1: Let $\mathcal{S}$ be an object schema with schema graph $G_{\mathcal{S}} = (V,E,l)$. An object functional dependency (OFD) of $\mathcal{S}$ is an edge- and node-labeled graph $f = (G_f, \nu_f)$ with the following properties:

(i) The OFD graph $G_f = (V_f, E_f, l_f)$ is an edge-labeled spanning tree of node set $V_f \subseteq V$ in $G_{\mathcal{S}}$ with $V_f \neq \varnothing$. l_f is the restriction of l to E_f.

(ii) $\nu_f : V_f \to \mathcal{D}_{\mathcal{S}} \times \mathcal{D}_{\mathcal{S}}$ is a partial node-labeling function such that for each $O \in V_f$ with ν_f defined and $\nu_f(O) = (\delta,\gamma)$ holds: $\delta,\gamma \in sets(O)$ and $\delta \cup \gamma \neq \varnothing$. ν_f has to be defined at least for every leaf node[1] of G_f.

O with $\nu_f(O) = (\delta,\gamma)$ and $\delta \neq \varnothing$ ($\gamma \neq \varnothing$) is called source (sink) object type of G_f or simply source (sink) type.

ν_f induces two partial functions $\nu_f^i := \Pi_i \circ \nu_f$, $i \in \{1,2\}$, with Π_i being the i-th projection. Each component function ν_f^i is undefined if the corresponding component of the ν_f-value is the empty set or if ν_f is undefined itself. □

[1] For a graph consisting of a single node, the node will be considered as a leaf, too.

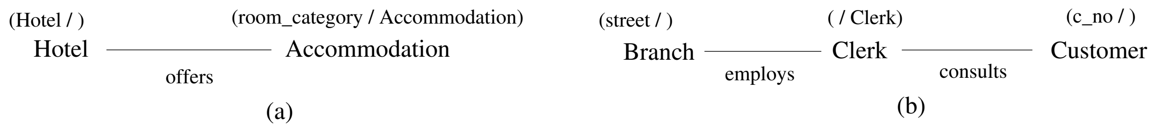


Figure 4: Examples of OFDs

The descriptions of node labels by ν_f or ν_f^1 and ν_f^2 are equivalent and will be used both. The specification of OFDs by spanning trees of nodes of the schema graph guarantees that there are no ambiguities with respect to the connections between sink and source types. The node-labeling function serves two purposes: it allows to separate object types actually used by an OFD, i. e. those determined by or determining an object type or values thereof, from those only connecting such types in G_f: for the latter ν_f is undefined. Moreover, ν_f denotes which parts of an object type are used to determine other types (first label component) or which are determined by other types (second component). Since an object type can be a source as well as sink type (e. g. object type *Accommodation* in Figure 1 (a)), the node labels are chosen as pairs. Leaf nodes are required to be labeled because only spanning trees of such node sets are to be considered where each leaf node contributes to the dependency, i. e. the type or attributes of it appear in the OFD. The OFDs corresponding to the examples from Figure 1 (a) and (b) are shown in Figure 4.

Each component of a node label consists either of an attribute set or of the object type itself. Attributes are not mixed together with the type since the object identifier uniquely determines an object and therefore its value, too.

The notion of an "object functional dependency" was used by Lee ([Lee95]) to denote functional dependencies restricted to a single class, where the object identifier is treated as a special attribute and may be used in dependencies.

A set oriented FD-like notation is often sufficient and more convenient to denote OFDs. It can be derived from the node labels of an OFD as follows:

Definition 3.2: Let $\mathcal{S}$ and $G_{\mathcal{S}}$ be as before and $f = (G_f, \nu_f)$ be an OFD of $\mathcal{S}$ with node set V_f.

$$\Delta \to \Gamma \text{ with } \Delta = \bigcup_{O \in V_f,\ \nu_f^1(O)\ defined} \{\nu_f^1(O)\},\ \Gamma = \bigcup_{O \in V_f,\ \nu_f^2(O)\ defined} \{\nu_f^2(O)\}$$

is the <u>set notation</u> of ν_f. Δ is called the <u>left side</u>, Γ the <u>right side</u> of the OFD f. $\delta \in \Delta$ is called <u>entry</u>. An object type O is <u>referred</u> to by the OFD (or <u>involved</u> in the OFD) iff O itself or any subset of its attribute set appears in the left or right side of f. f is called <u>type canonical</u> iff only one object type is involved in Γ. $\mathcal{N}(\Delta)$ ($\mathcal{N}(\Gamma)$) denotes the set of nodes of V_f, i. e. object types, involved in the left (right) side of f. $\mathcal{N}(\Delta \cup \Gamma)$ is a shorthand notation for $\mathcal{N}(\Delta) \cup \mathcal{N}(\Gamma)$ and denotes the set of all types involved in f. $O \in \mathcal{N}(\Delta)$ ($O \in \mathcal{N}(\Gamma)$) is called <u>$\Delta$-node</u> (<u>$\Gamma$-node</u>). The $\mathcal{N}(\cdot)$-notation will also be used for subsets Φ of $\mathcal{D}_{\mathcal{S}}$ analogously, denoting the set of object types belonging to the labels in Φ. $f : \Delta \xrightarrow{G_f} \Gamma$ (or, if G_f is uniquely determined by the underlying schema graph, simply $\Delta \to \Gamma$) will be written as a shorthand notation for an OFD $f = (G_f, \nu_f)$ with OFD graph G_f and set notation $\Delta \to \Gamma$ of ν_f. We occasionally omit set braces in order to simplify notation. □

Notation: For a given state, "left (right) side object" or "source (sink) object" will be used to denote objects belonging to the extension of a left (right) side object type. Furthermore, in the case of non type canonical OFDs, we will simply speak of right side objects when combinations of right side objects connected by link chains are meant.

Example 3.2: Figure 5 shows two OFDs f_1, f_2 of schema (b) from Figure 3, with $V_{f_1} = \{O_1, O_3, O_4, O_5\}$ and $V_{f_2} = \{O_1, O_2, O_3, O_4, O_5\}$. Both OFDs have the set notation $\{A, O_5, F\} \rightarrow \{O_4\}$.

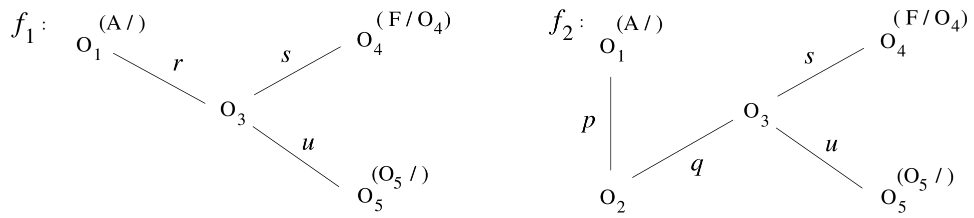


Figure 5: Examples of non local object dependencies

The OFDs from Figure 4 are written as $\{Hotel, room\_category\} \rightarrow \{Accommodation\}$ and $\{street, c\_no\} \rightarrow \{Clerk\}$ with graphs as given in Figure 4. □

The example demonstrates that the OFD graph is necessary to represent the subgraph of $G_{\mathcal{S}}$ referenced by an OFD f. The set notation alone is not sufficient. Every leaf node of G_f is an object type involved in f, whereas inner nodes are involved types or types on a path connecting two involved types.

From the set notation it can be seen whether an OFD is local (i. e. just one object type is involved in f) or non local (more than one type is involved in f). This corresponds to the notions of intra-object and inter-object constraints mentioned in [JQ92]. Both kinds of dependencies can be further divided into attribute based (left and right side of an OFD consisting of attribute sets only), object based (at least one object type appears in the left or right side) and pure object based (only object types involved in the left and right side) OFDs (cf. Figure 6). The OFDs f_1 and f_2 from the previous example are both non local object based OFDs.

4 The validation relation

Given a state $s(\mathcal{S})$ and an OFD $f : \Delta \xrightarrow{G_f} \Gamma$ of $\mathcal{S}$, it has to be defined what it means that f holds or does not hold in $s(\mathcal{S})$. This can be done based on a *validation relation* val_f for f with respect to $s(\mathcal{S})$. If f is a local OFD referring to object type O, the validation relation is determined

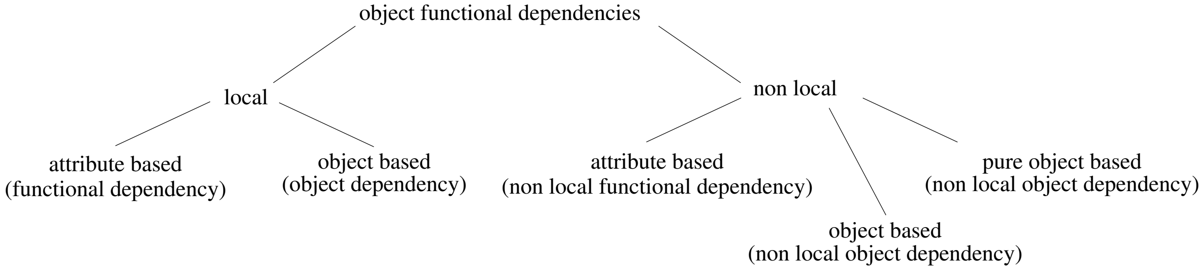


Figure 6: Types of object functional dependencies

uniquely by the relation representing $ext(O)$. If f is non local, such a relation is built by joining the relations corresponding to the relationships and object types of the paths connecting the referred types. To check a non local OFD with an object type O involved in the right side it has to be ensured that every object of $ext(O)$ appears in the validation relation if the requirement of surjective references ([ST93]) has to be fulfilled. Then the validity of an OFD under a given schema state can be defined similar to the validity of an FD with respect to a relation.

Depending on the extensions and the cardinalities of the relationships participating in G_f, there may exist right side objects having link chains to objects of only some of the types involved in Δ. It is not necessary that there is a link chain to at least one object of every type involved in Δ. The link tree corresponding to these chains will be called *partial linkage*. If link chains to objects of each of the left side types exist, we use the term *total linkage* or *linkage*, for short. Furthermore, there may exist right side objects having no links at all or only *insufficient link chains*. Insufficient link chains connect right side objects and objects of types being neither left side nor right side types and cannot be completed to a link chain connecting a right side and a left side object. If objects of a right side object type with only insufficient or no link chains exist, this might indicate that f is invalid. For the construction of the validation relation val_f this has two consequences: first, since partial linkages, insufficient link chains, or objects without any links may exist, val_f will be a partial relation in general. Second, the natural join cannot be used as operation to combine relations due to the well-known effect of 'dangling tuples'. To represent appropriately the part of $s(\mathcal{S})$ referenced by f the full outer join ([LP76]) has to be used, modified to operate on partial relations. Here the symbol $'-'$ represents missing links of an object. Therefore $'-'$ is a null value in the sense of 'value does not exist' and partial relations represent complete information about the underlying extensions. Moreover, the null value does not affect the evaluation of f on val_f: checking f is done by checking equality of attribute values of val_f. In this context a comparison $c = '-'$ can be evaluated to $false$ and $'-' = '-'$ to *true* for every domain value or object identifier c since two objects, one with, the other without link of the same relationship, can obviously be distinguished.

The null extended full outer join operation, modified to handle objects without links, i.e. dangling tuples which are undefined on the intersection attributes, is presented in the next definition.

Definition 4.1: Let R, S be partial relations over attribute sets α_R, α_S, respectively, with $\alpha_R \cap \alpha_S \neq \emptyset$. The <u>(null extended) full outer join</u> (FOJ) $R \overset{fo}{\bowtie} S$ of R and S is defined as

$$
\begin{aligned}
R \overset{fo}{\bowtie} S =_{df} \{t|t \text{ tuple over } \alpha_R \cup \alpha_S \wedge \\
&((t \text{ total on } \alpha_R \cap \alpha_S \wedge t|_{\alpha_R} \in R \wedge t|_{\alpha_S} \in S) \\
&\vee (t|_{\alpha_R} \in R \wedge t \text{ total on } \alpha_R \cap \alpha_S \wedge \neg(\exists t' \in S)(t|_{\alpha_R \cap \alpha_S} = t'|_{\alpha_R \cap \alpha_S}) \wedge t \text{ undefined on } \alpha_S \setminus \alpha_R) \\
&\vee (t|_{\alpha_S} \in S \wedge t \text{ total on } \alpha_R \cap \alpha_S \wedge \neg(\exists t' \in R)(t|_{\alpha_R \cap \alpha_S} = t'|_{\alpha_R \cap \alpha_S}) \wedge t \text{ undefined on } \alpha_R \setminus \alpha_S) \\
&\vee (t|_{\alpha_R} \in R \wedge t \text{ undefined on } \alpha_S) \\
&\vee (t|_{\alpha_S} \in S \wedge t \text{ undefined on } \alpha_R))\} \qquad \square
\end{aligned}
$$

By the first three join conditions the outer join is built for tuples total on the intersection of the attribute sets. By the last two conditions tuples undefined on the intersection are added to the result. This is sufficient for our purposes, since the set of join attributes will always be a singleton. The FOJ as defined above is associative and commutative. This does not hold if the operands are allowed to be relations over attribute sets with empty intersection.

Example 4.1: Consider the schema in Figure 7, consisting of object types O_1, O_2, O_3, O_4 with attribute sets $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, respectively, and relationships r_1, r_2, r_3. For the next

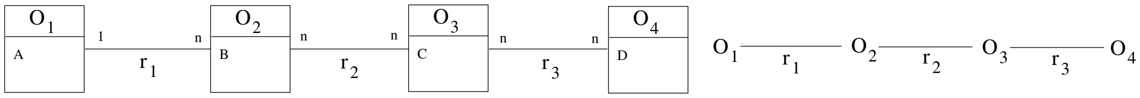


Figure 7: Simple schema and schema graph

examples let the following extensions be given:

$ext(O_1) = \{(1,[a]),(2,[a]),(3,[b])\}$, $ext(O_2) = \{(4,[c]),(5,[c]),(6,[a]),(7,[d]),(8,[d])\}$, $ext(O_3) = \{(9,[e]),(10,[f]),(11,[g])\}$, $ext(r_1) = \{(1,5),(1,6),(2,7),(3,8)\}$, $ext(r_2) = \{(6,9),(7,10)\}$

with relational representations derived appropriately. Application of the full outer join to $rel_{ext(O_3)}$ and $rel_{ext(r_2)}$ yields relation $R := rel_{ext(O_3)} \overset{fo}{\bowtie} rel_{ext(r_2)}$ as result:

$rel_{ext(O_3)}$

id_{O_3}	C
9	e
10	f
11	g

$rel_{ext(r_2)}$

id_{O_2}	id_{O_3}
6	9
7	10

R

id_{O_2}	id_{O_3}	C
6	9	e
7	10	f
–	11	g

If R is an intermediary relation to be used in further join operations, the fourth and fifth join condition ensure that partial tuples like $t = (-,11,g)$ are not lost. For example, when joining R with the relational representation of $ext(O_2)$, t would be represented in the result. Object $(3,[b])$ is an example of an O_1-object having only an insufficient link chain with respect to O_3-objects (assume an appropriate OFD to be given). $(4,[c])$ has no link chains at all. □

Using the FOJ, a relation for checking an OFD can be constructed:

Definition 4.2: Let $\mathcal{S}$ be an object schema, $s(\mathcal{S})$ be a state of $\mathcal{S}$ and $f : \{\delta_1,\ldots,\delta_k\} \xrightarrow{G_f} \{\gamma_1,\ldots,\gamma_l\}$ be an OFD with graph $G_f = (V_f, E_f, l_f)$ and node-labeling function ν_f. Let

$$\delta'_i := \begin{cases} \delta_i & \text{if } \delta_i \text{ is an attribute set} \\ \{id_O\} & \text{if } \delta_i = O, O \in OT_{\mathcal{S}} \end{cases}$$

and

$\Delta' := \bigcup_{i=1}^{k} \delta'_i$

($\gamma_j^{'}$, Γ' analogously).

The <u>validation relation</u> val_f of f under $s(\mathcal{S})$ is defined as follows:

(i) If f is a local OFD referring to object type O, val_f is the relational representation $rel_{ext(O)}$ of $ext(O)$.

(ii) If f is a non local OFD, val_f is obtained as follows: let $\{O_1,\ldots,O_l\} \subseteq V_f$ be the set of all sink object types and $\iota := \{id_{O_1},\ldots,id_{O_l}\}$ the set of their identifier attributes. For a node $O \in V_f$ let $\phi_O := \delta \cup \gamma \cup \{id_O\}$ if $\nu_f(O) = (\delta,\gamma)$ and $\phi_O := \{id_O\}$ if $\nu_f(O)$ is undefined.

(a) select a start node $O \in V_f$; $\mathcal{V} := rel_{ext(O)}[\phi_O]$
for all edges $e \in E_f$ incident on O with $l_f(e) = r$:
$\mathcal{V} := \mathcal{V} \overset{fo}{\bowtie} rel_{ext(r)}$; remove e from E_f

remove O from V_f
while $V_f \neq \emptyset$:
    select a node $O' \in V_f$ with $id_{O'} \in \alpha_{\mathcal{V}}$
    $\mathcal{V} := \mathcal{V} \overset{fo}{\bowtie} rel_{ext(O')}[\phi_{O'}]$
    for all edges $e' \in E_f$ incident on O' with $l_f(e') = r'$:
        $\mathcal{V} := \mathcal{V} \overset{fo}{\bowtie} rel_{ext(r')}$; remove e' from E_f
    remove O' from V_f

(b) (right side (rs)-normalization) If $\iota \neq \emptyset$, remove all tuples undefined on ι:
$\mathcal{V} := WNF(\mathcal{V}, \iota)$

(c) (subsumption (sub)-normalization) Remove all tuples subsumed by another tuple on the attributes belonging to the left and right side types of f:
val_f is a maximal subset of $\mathcal{V}$ such that no two tuples t and t', $t \neq t'$, exist in val_f with t' subsuming t on $\Delta' \cup \Gamma'$

Δ' is called set of $\{\delta_1, \ldots, \delta_k\}$-attributes or left side attributes, Γ' is called set of $\{\gamma_1, \ldots, \gamma_l\}$-attributes or right side attributes of val_f. □

Remark: If the rs-normalization results in a relation $\mathcal{V}$ with different tuples having the same values on attribute set $\Delta' \cup \Gamma'$, more than one relation can be obtained from $\mathcal{V}$ by applying sub-normalization. With respect to equality on $\Delta' \cup \Gamma'$ this set of relations forms an equivalence class and the application of sub-normalization corresponds to the selection of a representative.

Example 4.2: Let $f : \{A, O_3\} \xrightarrow{G_f} \{O_2\}$ be an OFD of the schema in Figure 7. Here the OFD graph G_f is induced by the schema graph (except for the node labels) and val_f can be computed by the FOJ sequence implied by G_f, when starting at node O_3:

$$(((rel_{ext(O_3)}[id_{O_3}] \overset{fo}{\bowtie} rel_{ext(r_2)}) \overset{fo}{\bowtie} rel_{ext(O_2)}[id_{O_2}]) \overset{fo}{\bowtie} rel_{ext(r_1)}) \overset{fo}{\bowtie} rel_{ext(O_1)}[id_{O_1}, A]$$

Using relation R (omitting attribute C) and the relational representations from Example 4.1 we get the following intermediary relations and finally the validation relation val_f.

$R' = R \overset{fo}{\bowtie} rel_{ext(O_2)}[id_{O_2}]$

id_{O_2}	id_{O_3}
6	9
7	10
5	–
8	–
4	–
–	11

$R'' = R' \overset{fo}{\bowtie} rel_{ext(r_1)}$

id_{O_1}	id_{O_2}	id_{O_3}
1	6	9
2	7	10
1	5	–
3	8	–
–	4	–
–	–	11

$\mathcal{V} = R'' \overset{fo}{\bowtie} rel_{ext(O_1)}[id_{O_1}, A]$

id_{O_1}	A	id_{O_2}	id_{O_3}
1	a	6	9
2	a	7	10
1	a	5	–
3	b	8	–
–	–	4	–
–	–	–	11

The first four tuples of $\mathcal{V}$ result from the natural join condition of the FOJ, the next one is a dangling tuple from $rel_{ext(O_2)}$ resulting from the third condition and represents an O_2-object without any links. The last tuple is included because of the fifth join condition. It would be of relevance, e. g. for checking an OFD $f' : \{O_1\} \to \{O_3\}$ with the same OFD graph as f. The validation relation val_f of f is obtained from $\mathcal{V}$ by applying normalization step (b), resulting in the deletion of the last tuple which is undefined on id_{O_2}. No tuples are deleted by sub-normalization in this example. □

In case of a type canonical OFD, the construction may start with the right side object type. For every join operation the set of join attributes consists of a single identifier attribute. The construction process guarantees that all right side objects of the given state, including all link chains to objects of any types appearing as nodes of G_f, are represented by the validation relation, with the exception of tuples deleted during rs- and sub-normalization. Obviously, tuples undefined on every attribute of val_f that belongs to a sink object type, represent link chains in which no object of a right side object type participates. By discarding them no information about the extensions of the sink object types is lost. For a sink object different linkages to combinations of source objects may exist, with one combination less specific than the other, i. e. two different tuples in val_f exist, having the same values on the right side attributes, and one tuple subsuming the other on the left side attributes. Such less specific combinations are eliminated by sub-normalization since they are subsumed by a more specific combination of entry values. This step may delete information about an object o of a sink type O if O is only involved at attribute level in the OFD, but not at type level. If an additional sink object with identical values on these attributes exists in the given state, the information about o might be discarded from the validation relation, because sub-normalization is done with respect to the attributes of the sink type, ignoring the identifier attribute. In this case the OFD specifies a dependency at value level and not all objects of $ext(O)$ have to be taken into account but only those having different values on the attributes of O specified in the OFD. Thus, concerning identification sub-normalization does not result in a loss of information because no value of the right side attributes occurring in only one tuple is discarded and the more exact combination of entry values is kept. Sub-normalization is necessary because subsumed tuples might interfere with the evaluation of f on val_f.

5 Object functional dependencies - semantics

Based on the validation relation different notions of validity can be given for an OFD with respect to a state of an object schema. These notions correspond to different views of identification. For example, in some situations only those objects of a type that are accessible by using certain paths may be of interest to a user (e.g. "all customers with orders that are not paid for yet") while in other situations not all of these paths may be considered as mandatory.

This kind of accessibility has not to be confused with the qualification of objects by conditions in query languages. Here we are interested in the unique access to single objects in a given set of objects by a fixed pattern.

5.1 Adopting FD semantics

The most obvious way to distinguish objects is to simply adopt the meaning of an FD $g : \beta \to \zeta$ for OFDs: each combination of values in the attributes of β in a given relation determines at most one ζ-combination, i. e. g is a function mapping β-combinations to ζ-combinations, with tuples being total. In the context of an OFD $f : \Delta \xrightarrow{G_f} \Gamma$ this means to look only at the value attributes and identifier attributes of val_f, belonging to Δ and Γ. The link chains between Δ- and Γ-objects are not considered although they are represented by val_f, too. This view corresponds to the following definition:

Definition 5.1: Let $\mathcal{S}$ be an object schema with state $s(\mathcal{S})$ and $f : \{\delta_1,\ldots,\delta_k\} \xrightarrow{G_f} \{\gamma_1,\ldots,\gamma_l\}$ be an OFD with validation relation val_f under $s(\mathcal{S})$ and Δ', Γ' as in Definition 4.2.
Let $\{id_{O_1},\ldots,id_{O_l}\}$ be the set of identifier attributes of the sink object types of f.

f is strongly satisfied by $s(\mathcal{S})$ iff the following conditions hold:

(i) $(\forall\, t,t' \in SNF(val_f,\Delta'))(t|_{\Delta'} = t'|_{\Delta'} \;\Rightarrow\; t|_{\Gamma'} = t'|_{\Gamma'})$

(ii) $val_f[\{id_{O_i}\}] = SNF(val_f,\Delta')[\{id_{O_i}\}]$ for each $i \in \{1,\ldots,l\}$

with $SNF(val_f,\Delta')$ denoting the strong null filter for val_f on attribute set Δ'. □

Condition (i) states that f induces a function, mapping each *total* combination of Δ'-values of val_f to exactly one Γ'-combination. Given values (objects, respectively) for each entry δ_i, at most one combination of right side objects or values is reached. Condition (ii) guarantees surjectivity (reachability) if all partial linkages are discarded. Thus, for each sink object at least one combination of source objects (values of objects, respectively) has to exist via which it can be accessed uniquely.

Example 5.1: (continuing Example 4.2) For $f : \{A,O_3\} \xrightarrow{G_f} \{O_2\}$ and val_f from Example 4.2 it has to be checked whether $t|_{\{A,id_{O_3}\}} = t'|_{\{A,id_{O_3}\}} \Rightarrow t|_{\{id_{O_2}\}} = t'|_{\{id_{O_2}\}}$ holds for every $t,t' \in SNF(val_f,\{A,id_{O_3}\})$. $SNF(val_f,\{A,id_{O_3}\})$ consists of tuples $(1,a,6,9)$ and $(2,a,7,10)$ which satisfy the condition. However, condition (ii) is violated (three tuples are eliminated by the strong null filter on $\{A,id_{O_3}\}$). Therefore f is not strongly satisfied by the given schema state. Consider another OFD $f' : \{O_2\} \xrightarrow{G_{f'}} \{A\}$, with respect to the same schema and state. The validation relation of f' consists of the first four tuples of $\mathcal{V}$ from Example 4.2 restricted to attributes id_{O_1}, A, and id_{O_2} ($val_{f'}$ in Figure 8). f' is strongly satisfied. OFD $f'' : \{B\} \xrightarrow{G_{f''}} \{A\}$, obtained by a left side change of f' to attribute level, is not strongly satisfied (cf. $val_{f''}$ in Figure 8). □

An OFD $f : \Delta \to \Gamma$ specifies a set of entries for accessing Γ-combinations. If an *entry value* is given for an entry $\delta \in \Delta$, it may select one (if δ is an object type) or possibly more (if δ is an attribute set) objects as *entry points* in the given state. Analogously a combination of entry values (short: *entry combination*) for Δ may select one or more sets of entry points, with each set corresponding to a linkage leading to a Γ-combination.

If f is strongly satisfied by a state and if a sink type O is involved in the OFDs right side at type level, this means that the O-objects can be distinguished taking only total entry combinations into account and using values and objects as entry values. Each O-object is reachable in this way, since f is surjective. It has to be noted that if O is involved in the right side of f only at

$val_{f'}$

id_{O_1}	A	id_{O_2}
1	a	6
2	a	7
1	a	5
3	b	8

$val_{f''}$

id_{O_1}	A	id_{O_2}	B
1	a	6	a
2	a	7	d
1	a	5	c
3	b	8	d

$val_{f'''}$

id_{O_1}	id_{O_2}	B	id_{O_3}	C
1	5	c	–	–
1	6	a	9	e
2	7	d	10	f
3	8	d	–	–

Figure 8: Validation relations of different OFDs under the same state

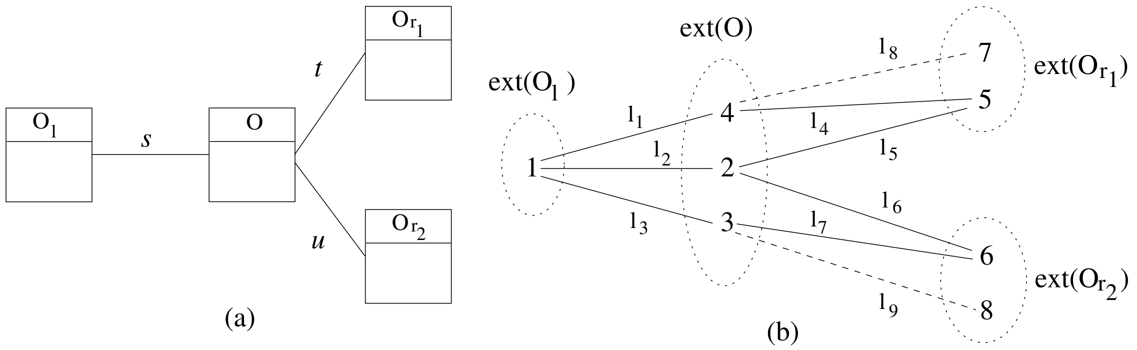


Figure 9: Fork schema with state

attribute level, surjectivity refers to the different attribute values, not to the set of all O-objects (cf. remarks on sub-normalization, page 12), and thus, strong satisfaction does not necessarily imply surjectivity at object level. In this case not every sink object may be reachable by using a total entry combination. But if such an object o exists and f is strongly satisfied, another O-object exists in the given state, reachable via a total entry combination and having the same values on the attributes appearing in f. This coincides with the view of OFDs as integrity constraints at object or value level: if an OFD is specified at value level (i. e. only attributes appear in the right side), different objects having identical values on the sink attributes need not to be distinguished since it is a coarser constraint than an OFD specified at object level.

The notion of identification defined by strong satisfaction follows closely the view of FDs. This becomes evident especially in the case of non type canonical OFDs.

5.2 Non type canonical OFDs and forkings

If a type canonical OFD is considered, i. e. an OFD with only one sink object type, different linkages with identical source objects (values, respectively) may exist for a single sink object. In this case a set of linkages may be determined by a combination of entry values, but they all lead to the same sink object. Thus, unique access to the object is guaranteed. However, in the case of non type canonical OFDs unique access to *combinations* of sink objects is needed in the context of strong satisfaction, and in contrast to type canonical OFDs, partial combinations may appear. Conflicts arise if different linkages with identical entry points exist, even if for each sink type the same object is involved in these linkages:

Consider object types O_l, O, O_{r_1}, O_{r_2} and relationships s, t, u between them as presented in Figure 9 (a), with relationships t, u being optional, and the simple extensions given by the graphical representation in Figure 9 (b). For clarity, only the types and identifiers are listed, no attributes or values. The solid lines represent links between objects. Furthermore, consider the OFD $f : \{O_l\} \xrightarrow{G_f} \{O_{r_1}, O_{r_2}\}$ with G_f as shown in Figure 9 (a), stating that O_{r_1}- and O_{r_2}-objects are distinguishable by their associated O_l-objects. This means that every, possibly partial, combination of O_{r_1}- and O_{r_2}-objects, connected by a link chain via an O-object under the given state, is uniquely determined by at least one O_l-object. For example, combination 5/6 is determined by O_l-object 1, using links l_2, l_5, l_6 and connecting O-object 2. However, with relationships t and u assumed to be optional, there may also exist link chains, such as 1 l_1 4 l_4 5 and 1 l_3 3 l_7 6, connecting an O_l-object with only one right side object. Here a single O_{r_1}-object (5) and a single

id_{O_l}	id_O	$id_{O_{r_1}}$	$id_{O_{r_2}}$	
1	4	5	–	∗
1	2	5	6	
1	3	–	6	∗

(a)

id_{O_l}	id_O	$id_{O_{r_1}}$	$id_{O_{r_2}}$
1	4	7	–
1	2	5	6
1	3	–	8

(b)

id_{O_l}	id_O	$id_{O_{r_1}}$	$id_{O_{r_2}}$
1	4	7	–
1	3	–	8

(c)

Figure 10: Validation relations of a non type canonical OFD with fork graph

O_{r_2}-object (6), as well as the combination of both, have to be identified by the same O_l-object (1). With respect to identification, this is not critical: starting with object 1, exactly one O_{r_1}- and one O_{r_2}-object can be reached using all link chains in the given state and a single linkage exists, providing access to both objects. These linkages are represented by the tuples shown in the relation from Figure 10 (a). Here, sub-normalization discards tuples (marked by $*$) because they are subsumed on the right side attributes. The validation relation for the given state consists of the total tuple only, representing the linkage that provides the "maximal information" about the access to sink objects 5 and 6 with object 1 as entry point.

Consider links l_4 and l_7 replaced by links l_8 and l_9 to additional new objects 7 and 8 (dashed lines in Figure 9 (b)). Again various combinations of sink objects can be reached from entry object 1, but now two different O_{r_1}-objects (O_{r_2}-objects) appear in the sink combinations, indicating that f does not hold. The validation relation for this state is shown in Figure 10 (b). The sub-normalization does not remove the two partial tuples in this case. However, for an OFD to be strongly satisfied, it is not sufficient that in the different linkages with identical entry combinations always the same object per sink type occurs: consider a state corresponding to the validation relation in Figure 10 (c) with links l_2, l_4, l_5, l_6, and l_7 missing in the state shown in Figure 9 (b). Here, only one object per sink type is reachable from entry object 1. Also OFDs $\{O_l\} \longrightarrow \{O_{r_1}\}$ and $\{O_l\} \longrightarrow \{O_{r_2}\}$ are strongly satisfied. But concerning strong satisfaction of f, the combinations of O_{r_1}- and O_{r_2}-objects have to be considered, corresponding to the functional view of an OFD: f maps O_l-objects to (possibly partial) pairs from $ext(O_{r_1}) \times ext(O_{r_2})$ and not to single O_{r_1}- and O_{r_2}-objects. Thus, f is not strongly satisfied with respect to this state, because two different sink combinations are reachable from one entry object and no "covering" linkage providing access to both objects 7 and 8 exists.

Thus, with respect to non type canonical OFDs, the notion of identification connected with strong satisfaction is restrictive in the sense that identification is not possible if more than one combination of sink objects is reachable from a given set of entry values, even if at most one object of each sink type does appear in all these combinations. For identification purposes there has to exist a sink combination reachable from the same entry points that combines the information of all these partial sink combinations.

The graph structure from which constellations with partial sink combinations as given in Figure 10 may arise corresponds to that of a *fork schema* or *fork graph*: at least three different object types (O_l,O_{r_1},O_{r_2} in Figure 9 (a)) have to be involved in f, with a minimum of two sink object types among them, such that not all three are nodes of a path in G_f. Therefore an additional type, the *fork type* (O), must exist, from which separate paths to the involved object types emerge. The graph in Figure 9 (a), represents the simplest constellation of types possibly leading to the conflicts shown in the validation relations of Figure 10. Furthermore, each of the paths leading from the fork type to the sink object types has to contain at least one optional relationship. The "handle" path of the fork graph connects the source type and the fork type, the "tine" paths lead from the fork type to the sink types. The involved object types do not have to be connected directly to the fork type, and more than one source type or further sink types may be involved in the OFD. But every OFD graph where linkages as presented in Figure 10 may arise in an associated state is non type canonical and contains a fork graph as subgraph. Thus, the existence of a fork type with the described properties concerning the positions of source and sink types is crucial.

If an OFD graph contains a fork graph, the OFD is non type canonical. The reversal does not hold in general, i. e. the class of OFDs with OFD graph containing a fork graph is a proper subclass of the class of non type canonical OFDs.

Remark: Here, as in the previous sections, we have argued primarily on the object level when investigating identification. Demanding the equality of values of objects is a much weaker condition and so all statements are still valid if a set of objects with identical values on the right side attribute sets of an OFD is considered instead of just a single object (analogously for the left side).

Whereas FDs, considered as functions on a (total) relation, are always surjective, this does not hold for non local OFDs in general: with respect to a path Π of an OFD, connecting a left side object type O_l and a right side type O_r, an O_r-object may have link chains to O_l-objects, insufficient link chains to objects of types on the path other than O_l, or even no link chains at all. In case of more than one left side object type also partial linkages of an O_r-object may appear. If it only has insufficient link chains, this means that it cannot be reached from any O_l-object by simply following a link chain, and thus, the OFD as a function mapping to O_r-objects is not surjective. In contrast to FDs this does not imply that the schema state is incomplete. However, in the sense of strong satisfaction, no value combination exists to distinguish such an O_r-object from others since totality on Δ' and the surjectivity property are required for this. Relaxation of these conditions leads to distinct forms of satisfaction which are of interest for the identification of objects. These will be discussed in the following sections.

5.3 Object centered versus link centered view

Strong satisfaction of an OFD guarantees totality on Δ and surjectivity, i. e. reachability from the total entry combinations. Giving up surjectivity corresponds to a view of identification, where we are only interested in such sink objects that can be reached by giving entry values for all Δ-elements and following the linkages selected by these values. Right side objects having only partial linkages to left side objects are discarded under this view, for they cannot be retrieved by giving a value combination for Δ. If all right side objects accessible in this way can be distinguished by looking at the Δ-values of their associated source objects, f will be called *weakly satisfied* by the schema state:

Definition 5.2: Let $\mathcal{S}$, $s(\mathcal{S})$, f, Δ', Γ', $\{id_{O_1},\ldots,id_{O_l}\}$ and val_f as in Definition 5.1. f is weakly satisfied by $s(\mathcal{S})$ iff condition (i) from Definition 5.1 holds. □

Example 5.2: (continuing Example 4.2) OFD $f : \{A, O_3\} \xrightarrow{G_f} \{O_2\}$ is weakly satisfied in the given state. $f'' : \{B\} \xrightarrow{G_{f''}} \{A\}$ from Example 5.1 is not weakly satisfied by the given state (cf. $val_{f''}$ from Figure 8). □

The validation relation as defined can be described as an object centered view of the state.

Every object of the sink types[2] of the underlying OFD is represented, including all links of it. Using only the relational representations of relationships to build the validation relation would lead to a more restrictive, link centered view. Objects without any link are ignored. If we try to construct such a validation relation by using the natural join, each partial linkage will be discarded, resulting in a total relation. The outer join, however, includes even "isolated" objects, i. e. objects without any link.

Weak satisfaction represents this link centered view. With respect to VBICs, weakly and strongly satisfied OFDs represent two different types of OFDs: when looking for VBICs usually only those criteria are of interest, that allow to access each object belonging to the extension of a considered type. Therefore in general a weakly satisfied OFD cannot be used as a VBIC. However, if an OFD is weakly satisfied, all those objects of the sink object type of an OFD are uniquely determined which are "visible" by using the elements of Δ as entries.

Our examples in Figure 1 (c) and Figure 2 show that totality on Δ can be too strong a condition if we look for possibilities to identify objects. Thus, partial linkages should be taken into account.

5.4 Partial linkages

In this section we will discuss the use of missing links and partial linkages (PLs) for the identification of objects. Null values in a validation relation can originate from these elements only. Objects with partial values, i. e. null values in an object value, are not considered here.

5.4.1 Navigational semantics

Consider an OFD $g : \{A,B\} \xrightarrow{G_g} \{O\}$ with A and B being attributes of object types O_1 and O_2, both directly connected to type O, and extensions given for these. g specifies (value based) entries that allow to retrieve objects of type O by navigating along the relationships connecting O_1, O_2 and O. We will denote this as *navigational semantics* of OFD g (cf. [Lie82]). Given a combination of values for A and B, and using G_g under the given state, at most one O-object is retrieved if g is valid. As described in Section 5.2 this navigational view does also fit for non type canonical OFDs, i. e. for combinations of sink objects.

Suppose we want to retrieve an O-object with A-value 1, B-value 2 and a second O-object with A-value 1 but no B-value, i. e. with no link to any O_2-object in the given state. The first can be accessed by starting with objects denoted by the A- and B-value, following the links of the selected O_1- and O_2-objects to an O-object if such exists for the given combination of entry values. For the second object this is not possible since no B-value exists. All O-objects with associated A-value 1 have to be selected instead, using only A as entry, and in a further step it has to be checked whether an object without a B-value exists among these. This is contrary to the navigational view of identification where values and objects are used to select entry points for traversing the relationships specified by the OFD graph. From this practical point of view, a missing link at object level can only be used in the additional "hidden" selection, but not as entry value. Thus, for the selection of a linkage (or a set of linkages) leading to a single sink object, missing links cannot be utilized accordingly.

[2] Or all different values of sink objects, if only attributes of a sink type occur in the OFDs right side.

Nevertheless, when investigating identification, missing links and PLs, and thus null values in the validation relation, arise naturally in the context of some modeling concepts, e. g. exclusive- or constrained relationships and inheritance: depending on the cardinality of the inheritance hierarchy an ancestor type O_1 usually identifies its inheriting subtypes O_2 and O_3. This can be specified by an OFD $\{O_1\} \xrightarrow{G} \{O_2, O_3\}$, where under a given state missing links of O_1-objects with respect to O_2- or O_3-objects may appear if the inheritance is optional. In the context of inheritance, missing links of the right side objects are also possible as shown by the example in Figure 2. This situation can be represented by an OFD $h : \{buyer\_no, company\_no\} \xrightarrow{G_h} \{Rental\ Institution\}$, where missing links of *Rental Institution*-objects with respect to the left side object types may appear. Here the notion of strong satisfaction would be too restrictive because the *buyer_no*/*company_no*-combination associated to a *Rental Institution*-object is always partial. This example also shows that the current definition of OFDs does not allow the exact specification of some kinds of VBICs. From the cardinality of the inheritance structure follows that missing links play no role for identification by h. Each *Rental Institution*-object is identified by either an *Investor*- or an *Administering Company*-object. This can only be captured by an (exclusive) disjunctive combination of the left side sets of h. In this paper, only conjunctions of the left side sets are considered for OFDs, similar to the definition of FDs. Exclusive-or like constraints demonstrate the necessity to consider missing links when investigating VBICs or OFDs as integrity constraints at schema level, even if they are not to be used for identification purposes.

5.4.2 Using partial linkages for identification

Partial linkages in the form of partial tuples of val_f can be used to distinguish objects. At object level missing links cannot be exploited in this way directly, because it is not possible to use them as entry values for the retrieval of objects (cf. Section 5.4.1). Taking advantage of partial linkages or missing links as they are represented by the validation relation, the following notions of *PL-satisfaction*, i. e. satisfaction considering partial linkages, can be derived from strong and weak satisfaction:

Definition 5.3: Let $\mathcal{S}$, $s(\mathcal{S})$, f, Δ', Γ', $\{id_{O_1}, \ldots, id_{O_l}\}$ and val_f be as in Definition 5.1.
f is <u>strongly PL-satisfied</u> by $s(\mathcal{S})$ iff the following conditions hold:

(i) $(\forall\, t, t' \in WNF(val_f, \Delta'))(t|_{\Delta'} = t'|_{\Delta'} \;\Rightarrow\; t|_{\Gamma'} = t'|_{\Gamma'})$

(ii) $val_f[\{id_{O_i}\}] = WNF(val_f, \Delta')[\{id_{O_i}\}]$ for each $i \in \{1, \ldots, l\}$

$WNF(val_f, \Delta')$ denotes the weak null filter for val_f on Δ'.
f is <u>weakly PL-satisfied</u> by $s(\mathcal{S})$ iff condition (i) holds. □

Strong PL-satisfaction represents a view of identification where at most one right side object can be reached, even if only for some of the entries objects or values exist. As mentioned before, for the check of condition (i) the null value will be treated as a constant, being different from all other domain values. A notion of identification in the sense of weak satisfaction, i. e. discarding surjectivity, is represented by weak PL-satisfaction.

Strong and weak PL-satisfaction exploit the use of partial linkages for the distinction of objects. As with strong and weak satisfaction, right side objects having only insufficient or no link chains to objects of source types are not considered since they are not reachable from any of the specified entries.

Example 5.3: (continuing Example 4.2) Because objects $o = (5,[c])$ and $o' = (8,[d])$ of $ext(O_2)$ have no links to an O_3-object, OFD $f : \{A, O_3\} \xrightarrow{G_f} \{O_2\}$ is weakly PL-satisfied by the given state. f is not strongly PL-satisfied due to object $o = (4,[c])$ having no links. OFD $f''' : \{B,C\} \xrightarrow{G_{f'''}} \{O_1\}$ is strongly PL-satisfied (cf. $val_{f'''}$ from Figure 8). □

Besides the problem with respect to the navigational point of view, the use of missing links for identification is limited, depending on the number of Δ-nodes. The maximum number of objects distinguishable by missing links also depends on the chosen type of identification, e.g. conjunctive or disjunctive combination of the object types involved in the left side of an OFD. The use of disjunctive combination allows for every state to distinguish right side objects by using only objects (or values) of one of the left side sets. This reduces the maximum number of objects distinguishable by missing links to 1.

5.5 Linkage conflicts

Further problems may arise when partial linkages or insufficient link chains are utilized for object identification. This is shown by the next example:

Example 5.4: Consider the schema from Figure 7 with extensions given by the following relational representations:

$rel_{ext(O_1)}$

id_{O_1}	A
1	a
2	b

$rel_{ext(r_1)}$

id_{O_1}	id_{O_2}
1	3
2	5

$rel_{ext(O_2)}$

id_{O_2}	B
3	a
4	c
5	c
6	b

$rel_{ext(r_2)}$

id_{O_2}	id_{O_3}
3	7
4	7
5	8
5	9
6	8

$rel_{ext(O_3)}$

id_{O_3}	C
7	e
8	e
9	d

$rel_{ext(r_3)}$

id_{O_3}	id_{O_4}
7	10
8	10
9	11

$rel_{ext(O_4)}$

id_{O_4}	D
10	f
11	g

Under this state the relation obtained for OFD $f : \{O_1, O_4\} \xrightarrow{G_f} \{O_3\}$ prior to the application of sub-normalization contains three total and two partial tuples, corresponding to the linkages of O_3-objects to O_1- and O_4-objects (cf. relation $\mathcal{V}$ in Figure 11). The partial tuples, marked with $*$ in $\mathcal{V}$, would interfere with the way f is evaluated but they are deleted by sub-normalization since subsuming tuples exist. □

Unfortunately, this normalization step may delete information from the validation relation by which objects could be distinguished, because in some cases this is possible only by using PLs:

Example 5.5: Consider extensions for the relationships and object types of the schema from Figure 7, leading to the relation $\mathcal{V}'$ shown in Figure 11 before sub-normalization (using OFD $f : \{O_1, O_4\} \xrightarrow{G_f} \{O_3\}$). O_3-objects $(7,[\ldots])$ and $(8,[\ldots])$ have link chains to the same O_1- and O_4-objects and therefore cannot be distinguished by them. But using the PL for object $(7,[\ldots])$, the two objects can be separated by their associated id_{O_1}-values. During sub-normalization the second instead of the first tuple is deleted, resulting in a validation relation by which f is not satisfied, although the two O_3-objects obviously can be distinguished via their link chains to O_1-objects. □

$\mathcal{V}$

id_{O_1}	id_{O_4}	id_{O_2}	id_{O_3}	
1	10	3	7	
–	10	4	7	*
2	10	5	8	
–	10	6	8	*
2	11	5	9	

$\mathcal{V}'$

id_{O_1}	id_{O_4}	id_{O_2}	id_{O_3}
1	10	3	7
–	10	4	7
1	10	5	8
1	11	5	9

$\mathcal{V}''$

id_{O_1}	id_{O_4}	id_{O_2}	id_{O_3}
1	10	3	7
–	10	4	7
1	10	5	8
–	10	6	8
2	10	13	16
2	10	14	8
12	10	15	7
12	11	15	9

val_g

id_{O_1}	id_{O_2}	id_{O_3}
1	3	7
1	4	8
–	5	7
–	5	8

$\{O_1, O_2\} \xrightarrow{G_g} \{O_3\}$

$\{O_1, O_4\} \xrightarrow{G_f} \{O_3\}$

Figure 11: Linkage conflicts

The kind of effect described in Example 5.5 arises only if a subsuming tuple in the validation relation exists and PL-satisfaction is considered. We will call constellations as shown in Figure 11, where identification of objects is possible but disturbed by the existence of multiple (partial or total) linkages for a sink object, *linkage conflicts*. $\mathcal{V}$, $\mathcal{V}'$ and $\mathcal{V}''$ represent different kinds of linkage conflicts. Some disappear by sub-normalization (cf. $\mathcal{V}$). In other cases sub-normalization may remove tuples necessary for identification (cf. $\mathcal{V}'$) or it has no effect at all since the linkage conflict arises from total linkages (cf. $\mathcal{V}''$) or additional PLs as shown in the next example.

Example 5.6: Consider OFD $g : \{O_1, O_2\} \xrightarrow{G_g} \{O_3\}$ with graph induced by the schema from Figure 7 and a schema state leading to the validation relation val_g as shown in Figure 11. Tuples $(-,5,7)$ and $(-,5,8)$ are not removed by sub-normalization because no subsuming tuples on $\{id_{O_1}, id_{O_2}, id_{O_3}\}$ exist. Both notions of identification taking PLs into account are affected by this. □

To avoid linkage conflicts involving PLs it would be necessary to decide which PLs are needed for identification. For a small number of objects this seems feasible, but it becomes more difficult with an increasing number of objects in a given state. Furthermore, as $\mathcal{V}''$ already indicates, the problem is not restricted to PLs. Linkage conflicts may also arise if only total linkages appear in the state: for example, replace all null values in the relations from Figure 11 by a new object identifier to obtain states having total linkages only.

5.6 Comparing the notions of identification

From the definitions of satisfaction follows directly:

Lemma 5.1: Let $\mathcal{S}$ be an object schema with state $s(\mathcal{S})$ and $f : \Delta \xrightarrow{G_f} \Gamma$ be an OFD. If f is strongly satisfied (strongly PL-satisfied) under $s(\mathcal{S})$, then f is weakly satisfied (weakly PL-satisfied) under $s(\mathcal{S})$. □

For a given relation R over attribute set α_R and $\beta \subseteq \alpha_R$ the strong null filter $SNF(R, \beta)$ is always a subset of the weak null filter $WNF(R, \beta)$. Thus, we have:

Lemma 5.2: Let $\mathcal{S}$ be an object schema with state $s(\mathcal{S})$ and $f : \Delta \xrightarrow{G_f} \Gamma$ be an OFD. If f is weakly PL-satisfied under $s(\mathcal{S})$, then f is weakly satisfied under $s(\mathcal{S})$. □

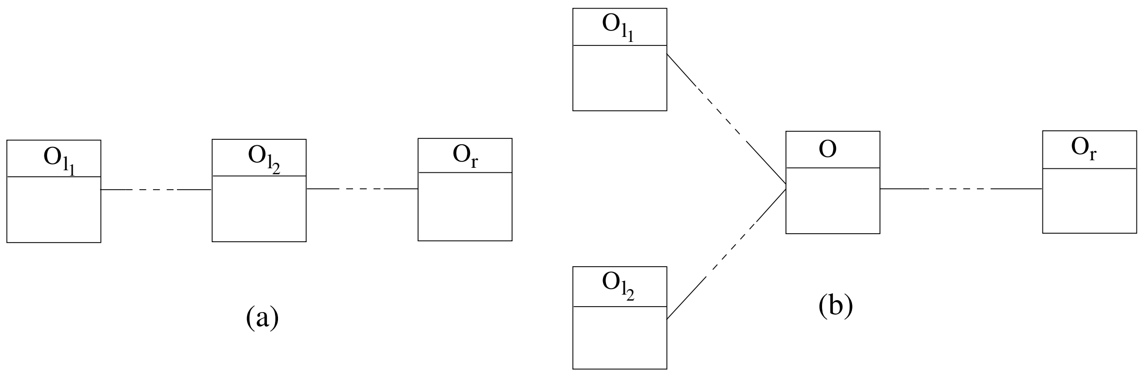


Figure 12: Shielding and converge constellation

In general the reversals of both lemmas do not hold. The notions of strong satisfaction and strong PL-satisfaction have to be examined more thoroughly. Whereas it is obvious that strong PL-satisfaction does not imply strong satisfaction, the navigational semantics of object identification suggests the opposite implication to hold as well: if in a given state an object is reachable by using an entry value for every specified entry and following the link chains induced, then this object is still reachable via the same combination of values if any additional PLs are considered, too. The reason for the implication not to hold are *PL conflicts*, linkage conflicts as shown in Example 5.6, which solely arise from additional PLs (cf. val_g in Figure 11).

By applying the null filter when checking for strong satisfaction not only the use of PLs for identification is prevented but also PL conflicts are discarded, i. e. the view of identification represented by strong satisfaction ignores PL conflicts. If all right side objects are distinguishable by total linkages and only PLs not leading to PL conflicts exist in the given state, strong PL-satisfaction is implied by strong satisfaction. Therefore criteria for the possible appearance of PL conflicts depending on the graph structure of an OFD are of interest.

Lemma 5.3: Let $\mathcal{S}$ be an object schema with state $s(\mathcal{S})$ and $f : \Delta \xrightarrow{G_f} \Gamma$ be an OFD, such that val_f contains no PL conflict. Then f is strongly PL-satisfied under $s(\mathcal{S})$ if f is strongly satisfied under $s(\mathcal{S})$. □

If some arrangements of source and sink nodes do not appear in an OFD graph, PL conflicts cannot arise. Obviously this is the case if only one type is involved in the left side of an OFD. Null values in an object value are not considered here, therefore in this case only tuples total or undefined on the left side attributes appear in the validation relation. The latter are not taken into account for identification. This leads to a first restricted criterion:

Lemma 5.4: Let $\mathcal{S}$ be an object schema with state $s(\mathcal{S})$, and $f : \Delta \xrightarrow{G_f} \Gamma$ be an OFD with at most one object type involved in Δ. Then val_f contains no PL conflict. □

Less restrictive criteria for source types of an OFD f are needed which exclude the existence of PL conflicts. Under two constellations PL conflicts may arise in a state: *shieldings* and *converges* with respect to sink types. A source object type O_{l_1} is *shielded* by another source type O_{l_2} if a path in G_f exists, connecting O_{l_1} and a sink object type O_r via node O_{l_2} (see Figure 12 (a)). For example, consider the graph from Figure 7 as OFD graph of $f : \{O_1, O_2\} \xrightarrow{G_f} \{O_4\}$. O_1 is shielded by O_2 with regard to O_4. From shieldings PL conflicts may arise since different O_r-objects can be connected to the same O_{l_2}-object with the O_{l_2}-object having no link to an O_{l_1}-object in a given state.

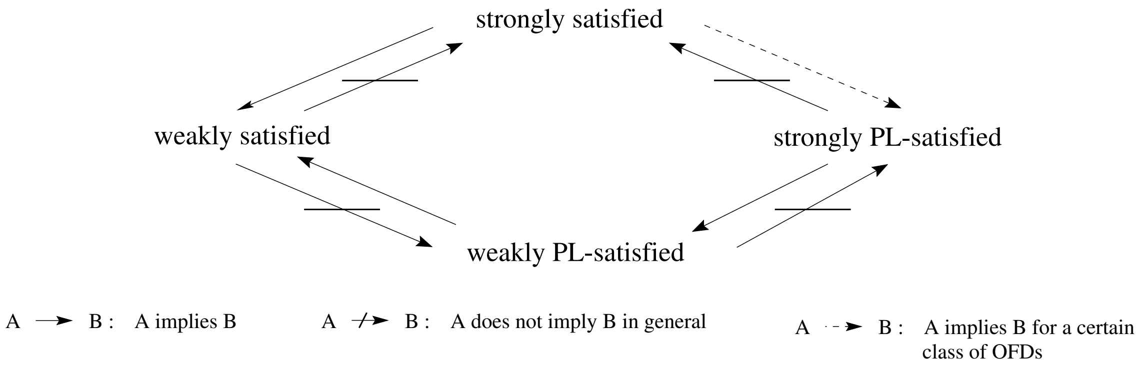


Figure 13: Implication relationships between the notions of identification

A *converge* can be regarded as a fork graph with source and sink nodes "inverted". In a converge constellation two source types O_{l_1} and O_{l_2} have a node O in common on their paths to a sink type O_r and there is no single path connecting O_{l_1}, O_{l_2} and O_r (see Figure 12 (b)). For example, consider the OFD graph G_f induced by nodes O_1, O_3, O_4 and O_5 of Figure 3 (b) and f : $\{O_1, O_4\} \xrightarrow{G_f} \{O_5\}$. Types O_1 and O_4 are a converge with respect to O_5. Converges may lead to PL conflicts in the following way: two O_r-objects can be connected to one O_{l_2}-object, but for the O-object involved in the link chains there may exist no chain to an O_{l_1}-object.

Shieldings and converges do not appear if every path of an OFD graph connecting two source types contains a sink node. We will call this condition *source path condition*. In this case a situation like that in Example 5.6, with right side object $(7, \ldots)$ having partial and total linkages (as represented by tuples $(-, 5, 7)$ and $(1, 3, 7)$) could not occur: $(7, \ldots)$ has a link chain to two O_2-objects and one O_1-object. If O_3 were the connecting node between source types O_1 and O_2, no PLs of $(7, \ldots)$ could exist. Therefore, if the source path condition is satisfied, a sink object o either has no link chain to any object of a source type O_l, i. e. every linkage of o is partial, or it has at least one such chain and no PLs with respect to O_l appear in the given state. The sink object type acts as a "cartesian product node", combining all link chains of a sink object to source objects to other link chains. The same does hold in the case of a converge. Following from this, no two tuples representing link chains of the same sink object, one being total, the other undefined on the same left side attribute can exist in the validation relation. As a consequence, no PL conflicts can arise. This is summarized by the next lemma:

Lemma 5.5: Let $\mathcal{S}$ be an object schema with state $s(\mathcal{S})$ and $f : \Delta \xrightarrow{G_f} \Gamma$ be an OFD. Let every path in G_f connecting two source nodes satisfy the source path condition. Then val_f contains no PL conflict. □

This lemma gives a more general criterion than Lemma 5.4. Strong PL-satisfaction follows from strong satisfaction for an OFD f if the paths connecting the sink object type O with the source types are disjoint, except for the node O itself. In this case, source types appear only as leafs of G_f without any converges. If non-leaf source types exist, additional sink nodes do separate them. Informally speaking, under the source path condition an object type can be determined by any set of types of the underlying schema graph $G_{\mathcal{S}}$. The "environment" of O in $G_{\mathcal{S}}$ contributing to the identification is not limited as far as the number of source types and the length of connecting paths in G_f are concerned. The implication relationships of the different notions of satisfaction are summarized in Figure 13.

Additional constraints of the object schema like cardinality restrictions have not been taken into account here. Integrating such schema information helps to avoid PL conflicts even if the source path condition is violated. For example, if all relationships on the path to a shielded object type are mandatory, no PL conflicts can result from a shielding.

Note that general PL conflicts were not considered here, but emphasis was on conflicts which are of relevance in the context of strong satisfaction and strong PL-satisfaction. The former are a special case of linkage conflicts caused by total linkages as shown in Figure 11 by relation $\mathcal{V}''$ or $\mathcal{V}'''$. To solve such conflicts other notions of identification which allow to select entries for each object have to be introduced.

5.7 OFDs as VBICs

Using OFDs we can formalize a VBIC for an object type O as an OFD f where O is element of the right side of f and no type occurs in the left side of f:

Definition 5.4: An OFD $f : \Delta \xrightarrow{G_f} \{O\}$ is called an identification criterion for an object type O of schema $\mathcal{S}$. A value based identification criterion (VBIC) for O is an identification criterion $f : \Delta \xrightarrow{G_f} \{O\}$ for O with $\delta \cap OT_{\mathcal{S}} = \varnothing$ for each $\delta \in \Delta$. □

Whereas an identification criterion specifies a set of entries possibly containing object types, a VBIC offers access to objects only by values, i. e. a pure value based replacement for the object identifier.

If VBICs shall be minimal like keys no proper subset Δ' of the left side Δ may exist, such that $f' : \Delta' \xrightarrow{G_{f'}} \{O\}$ holds and each $\delta \in \Delta$ has to be minimal, too. If no pure VBIC Δ can be found, i. e. Δ contains not only attribute sets but also object types, a different notion of minimality seems to be advisable since in this case a set Δ_1, containing less object types than a set Δ_2, might be preferable as set of entries, even if $|\Delta_1| > |\Delta_2|$ holds.

The different notions of identification have to be carefully taken into account when considering VBICs for an object type O since access to each object of O is usually desired. This is not guaranteed in the case of weak (PL-) satisfaction. Furthermore, as with functional dependencies, inference rules may be necessary for the derivation of VBICs. This matter will be addressed in Section 6.

6 Inference rules

OFDs may be used to specify dependencies between object types and/or attributes in the design phase of a database schema. As with FDs usually not all valid dependencies have to be given by the designer. Some dependencies are trivial or they follow from specified dependencies. In order to derive such dependencies inference rules are needed. In this section we will elaborate how rules similar to relational inference rules for FDs (reflexivity, decomposition, augmentation, union, transitivity and pseudotransitivity, cf. [Ull88]) are applicable to OFDs. As we shall see, additional rules are necessary in order to guarantee completeness of rule sets.

For a generalization of relational rules two approaches are conceivable: on the one hand, a notion of satisfaction, e. g. strong PL-satisfaction, can be chosen and the inference rules can

be modified in such a way that from a PL-satisfied OFD only OFDs are derivable which are PL-satisfied too. Depending on the choice of semantics, this may lead to strong restrictions of the OFDs involved in the derivation process and to additional requirements for the inference rules, respectively. On the other hand, relational inference rules may be adopted directly to OFDs, imposing as few as possible restrictions on the involved OFDs. With this proceeding a relaxation of the semantics may be necessary.
When considering rules for non local OFDs, we will follow the second approach, adopting the rules for strongly satisfied OFDs.

Inference rules are of interest for determining VBICs since the designer is not obliged to specify VBICs directly for each object type of the schema if VBICs can be derived automatically (cf. the derivation of candidate keys for a relational schema).

6.1 Adopting relational "axioms"

FDs are specified on an attribute set of a relation type. Their satisfaction is defined with respect to a single relation of that type. For an OFD f satisfaction is defined regarding a relation type specific to f. Two OFDs different by graph or labels are evaluated over validation relations with different sets of attributes in general. Thus, the schema state can serve as a common basis for checking them.

For a derived OFD a graph together with node labels has to be determined based on the initial OFDs. In case of a non local OFD the derived graph has to comply with the definition of an OFD graph (tree structure with all the leaf nodes being labeled). For local OFDs referring to the same object type relational inference rules are obviously applicable. Only minor syntactic modifications of the rules are necessary since FDs are defined over attribute sets, whereas OFDs are defined over sets of attribute and object type sets. The graph of a local OFD consists of a single node and remains unchanged for derived OFDs.

During the construction of new (local or non local) OFDs it might be necessary to combine labels of a node appearing in different OFD graphs to a new one. For this, a "combining operation" of labels and sets of labels is needed. If two labels belonging to the same node in different graphs are to be combined, and if the labels are attribute sets, the new label is obtained by building the union of the label components. Equality of objects is a stronger notion than the equality of attribute values. Therefore, mixing of attributes and object types should be avoided. An attribute set is ignored if one of the operands is an object type:

Definition 6.1: Let $\mathcal{S}$ be an object schema, $O \in OS_{\mathcal{S}}$, and $\delta, \gamma \in sets(O)$. The <u>label combination</u> $\delta \uplus \gamma$ of δ and γ is defined as

$$\delta \uplus \gamma = \begin{cases} O & \text{if } \delta = O \text{ or } \gamma = O \\ \delta \cup \gamma & \text{otherwise} \end{cases}$$

Let $\Delta, \Gamma \subseteq \mathcal{D}_{\mathcal{S}}$ be two sets such that no two different sets $\delta_1, \delta_2 \in \Delta$ or $\gamma_1, \gamma_2 \in \Gamma$ refer to the same object type. The <u>label combination</u> $\Delta \uplus \Gamma$ of Δ and Γ is defined as

$$\begin{aligned} \Delta \uplus \Gamma = \; & \{\delta \mid \delta \in \Delta \wedge \delta \in sets(O) \wedge O \notin \mathcal{N}(\Gamma)\} \\ & \cup \{\gamma \mid \gamma \in \Gamma \wedge \gamma \in sets(O) \wedge O \notin \mathcal{N}(\Delta)\} \\ & \cup \{\delta \uplus \gamma \mid \delta \in \Delta \wedge \gamma \in \Gamma \wedge \delta, \gamma \in sets(O)\} \end{aligned}$$

□

Example 6.1: Let O_1, O_2, and O_3 be object types with attribute sets $attr(O_1) = \{A,B,C,D\}$, $attr(O_2) = \{E,F,G\}$, $attr(O_3) = \{H,I\}$ and $\delta_1 = \{A,C\}$, $\gamma_1 = \{A,D\}$, $\delta_2 = \{E,F\}$, $\gamma_2 = \{O_2\}$, $\delta_3 = \{H\}$, and $\Delta = \{\delta_1,\delta_2,\delta_3\}$, $\Gamma = \{\gamma_1,\gamma_2\}$. Then we get $\delta_1 \uplus \gamma_1 = \{A,C,D\}$, $\delta_2 \uplus \gamma_2 = \{O_2\}$, and $\Delta \uplus \Gamma = \{\{A,C,D\},\{O_2\},\{H\}\}$. □

6.2 Rules for local OFDs

The modification of the node labels is straightforward when new local OFDs are derived. This shows the next lemma.

Lemma 6.1: Let $\mathcal{S}$ be an object schema with state $s(\mathcal{S})$, $O \in OT_{\mathcal{S}}$, $\delta,\gamma,\varepsilon \in sets(O)$ and $G = (\{O\},\varnothing,l_f)$ the OFD graph consisting of the single node O.

(Reflexivity) Let $\delta \in sets^+(O)$. Then $f : \{\delta\} \xrightarrow{G} \{\delta\}$ is strongly satisfied under $s(\mathcal{S})$.

(Decomposition) Let $f : \Delta \xrightarrow{G} \Gamma$, $\nu_f(O) = (\delta,\gamma)$ be strongly satisfied under $s(\mathcal{S})$. Then $g : \Delta \xrightarrow{G} \Gamma'$ with $\nu_g(O) = (\delta,\gamma')$, $\gamma' \subseteq \gamma$, and $\delta \in sets^+(O)$ or $\gamma' \in sets^+(O)$ is strongly satisfied under $s(\mathcal{S})$.

(Augmentation) Let $f : \Delta \xrightarrow{G} \Gamma$ be strongly satisfied under $s(\mathcal{S})$. Then $g : \Delta \uplus \{\varepsilon\} \xrightarrow{G} \Gamma \uplus \{\varepsilon\}$ is strongly satisfied under $s(\mathcal{S})$.

(Union) Let $f_1 : \Delta \xrightarrow{G} \Gamma_1$ and $f_2 : \Delta \xrightarrow{G} \Gamma_2$ be strongly satisfied under $s(\mathcal{S})$. Then $g : \Delta \xrightarrow{G} \Gamma_1 \uplus \Gamma_2$ is strongly satisfied under $s(\mathcal{S})$.

(Transitivity) Let $f_1 : \Delta \xrightarrow{G} \Gamma$, $f_2 : \Gamma \xrightarrow{G} \Phi$ be strongly satisfied under $s(\mathcal{S})$ and $\Delta \neq \varnothing$ or $\Phi \neq \varnothing$. Then $g : \Delta \xrightarrow{G} \Phi$ is strongly satisfied under $s(\mathcal{S})$.

(Pseudotransitivity) Let $f_1 : \Delta \xrightarrow{G} \Gamma$, $f_2 : \Gamma \uplus \{\gamma\} \xrightarrow{G} \Phi$ be strongly satisfied under $s(\mathcal{S})$ and $\Delta \neq \varnothing$ or $\Phi \neq \varnothing$ or $\gamma \in sets^+(O)$. Then $g : \Delta \uplus \{\gamma\} \xrightarrow{G} \Phi$ is strongly satisfied under $s(\mathcal{S})$.

□

For local OFDs, the left and right sides are singletons or one side is empty. This has to be taken into account for the construction of each derived OFD. The OFD graph, consisting of a single node, remains unchanged. Soundness of the rules for local OFDs follows obviously from the soundness of the relational rules.

Corollary 6.1: The rules for local OFDs given in Lemma 6.1 are also valid under weak satisfaction, strong PL-satisfaction and weak PL-satisfaction. □

6.3 Rules for non local OFDs

In contrast to local OFDs, inference rules for FDs cannot be adopted directly for non local OFDs since the OFD graphs, arbitrary spanning trees of a subgraph of the schema graph, have to be taken into account. Corresponding rules are only valid under certain restrictions for OFD graphs: if OFD $g : \Delta' \xrightarrow{G_g} \Gamma'$ is derived from $f : \Delta \xrightarrow{G_f} \Gamma$, object types occurring in both OFD graphs have to be connected in G_g in the same way as in G_f. This requirement is necessary in order to guarantee semantic compatibility of the OFDs involved. Example 3.1 shows that this is essential if the validity of g is to be inferred from the validity of f under the same schema state. This compatibility can be formalized by "V'-trees", spanning trees induced by a node set $V' \subseteq V$ of an OFD-graph $G_f = (V_f, E_f, l_f)$[3].

Among the given notions of satisfaction, strong satisfaction represents the most restrictive view of identification. It is closely related to FDs and thus, inference rules will be investigated for non local OFDs with respect to this semantics. We will show that strong satisfaction cannot always be guaranteed for derived OFDs.

In the following let $\mathcal{S}$ be an object schema with state $s(\mathcal{S})$ and $\Delta, \Gamma, \Phi \subseteq \mathcal{D}_{\mathcal{S}}$. If not stated otherwise, we assume an OFD graph $G_f = (V_f, E_f, l_f)$ and a node-labeling function ν_f to be given for each OFD f. val_f denotes the validation relation of f under $s(\mathcal{S})$.

Reflexivity Rule

> Let $f : \Delta \xrightarrow{G_f} \Delta$, $\Delta \neq \emptyset$, be an OFD with G_f being an $\mathcal{N}(\Delta)$-tree of $G_{\mathcal{S}}$. Then f is weakly satisfied.

Here, left and right side of f refer to the same types and attributes, and the sets of left and right side attributes are identical. Therefore f is weakly satisfied. However, strong satisfaction cannot be expected in general for the derived OFD if optional relationships are involved in the OFD graph. This is illustrated by the following example:

Example 6.2: Consider the schema from Figure 7, restricted to object types O_1, O_2, and O_3 and relationships r_1 and r_2. Assume a state with the following simple extensions to be given:

$$\begin{array}{ll} ext(O_1) = \{(1,[a]),(2,[a])\} & ext(r_1) = \{(1,3),(2,4)\} \\ ext(O_2) = \{(3,[b]),(4,[b])\} & ext(r_2) = \{(4,5)\} \\ ext(O_3) = \{(5,[c]),(6,[c'])\} & \end{array}$$

We get the following validation relation for the OFD $g : \{A, C\} \longrightarrow \{A, C\}$:

val_g

id_{O_1}	A	id_{O_2}	id_{O_3}	C
1	a	3	–	–
2	a	4	5	c
–	–	–	6	c'

It is easy to see that OFD g is not strongly satisfied, because the null filter on attributes A and C removes information about object $(1,[a])$ of O_1 and object $(6,[c'])$ of O_3. □

As demonstrated in the example, surjectivity is a crucial property for a non local OFD derived by an inference rule if strong satisfaction is required. If an OFD involves optional relationships, this condition may be violated in a given database state. The impact of the surjectivity property on the satisfaction of a derived OFD has also to be taken into account for the following rules.

[3] For graphtheoretic terms see the Appendix.

Decomposition Rule

> Let $f : \Delta \xrightarrow{G_f} \Gamma$ be an OFD where $\Gamma = \{\gamma_1, \dots, \gamma_l\}$, $l \geq 1$; let $\overline{\Gamma} = \{\overline{\gamma_1}, \dots, \overline{\gamma_l}\}$ with
>
> $$\overline{\gamma_i} := \begin{cases} \gamma & \gamma \subseteq \gamma_i, \gamma \neq \emptyset, \text{if } \gamma_i \text{ is an attribute set} \\ \gamma_i & \text{otherwise} \end{cases} \qquad \text{for } i \in \{1, \dots, l\}.$$
>
> If f is strongly satisfied and $f' : \Delta \xrightarrow{G_{f'}} \Phi$, $\Phi \subseteq \overline{\Gamma}$, $\Delta \neq \emptyset$ or $\Phi \neq \emptyset$, is the OFD with $G_{f'}$ the $\mathcal{N}(\Delta \cup \Phi)$-tree of G_f, then f' is strongly satisfied.

The given rule allows decomposition by omitting label sets from Γ as well as by omitting some attributes in any of the label sets. The new graph $G_{f'}$ is uniquely determined by the given OFD graph G_f and set Φ. The $\mathcal{N}(\Delta)$-trees of G_f and $G_{f'}$ are identical, and the two OFD graphs differ only if the restriction of Γ to Φ discards a Γ-node that is not also a Δ-node. As a consequence, $val_{f'}$ projected onto the set of Δ- and Φ-attributes is a subset of val_f projected onto the same set. From $\Phi \subseteq \overline{\Gamma}$ and the construction of $\overline{\Gamma}$ follows that f' is strongly satisfied if f is strongly satisfied.

Augmentation Rule

> Let $f : \Delta \xrightarrow{G_f} \Gamma$ be an OFD and $O \in OT_{\mathcal{S}}$, $\varepsilon \in sets(O)$, $f' : \Delta \uplus \{\varepsilon\} \xrightarrow{G_{f'}} \Gamma \uplus \{\varepsilon\}$ an OFD such that G_f is the $\mathcal{N}(\Delta \cup \Gamma)$-tree of $G_{f'}$ and
>
> $$\nu_{f'}(O') := \begin{cases} \nu_f(O') & \text{if } O' \in V_f \setminus \{O\} \\ (\varepsilon, \varepsilon) & \text{if } O' = O, O \notin V_f \text{ or } O' = O, O \in V_f, \\ & \quad \nu_f(O) \text{ undefined} \\ (\delta \uplus \varepsilon, \gamma \uplus \varepsilon) & \text{if } O' = O, O \in V_f \text{ and } \nu_f(O) = (\delta, \gamma) \\ \text{undefined} & \text{if } O' \in V_{f'} \setminus (V_f \cup \{O\}) \end{cases}$$
>
> If f is weakly satisfied, then f' is weakly satisfied.

If f is satisfied under $s(\mathcal{S})$ with $O \in V_f$, G_f and $G_{f'}$ are identical graphs. Only the node label for O has to be adjusted. The attribute sets α_{val_f} and $\alpha_{val_{f'}}$ differ only if $\varepsilon \subseteq attr(O)$, otherwise they are identical. Since ε is added to both sides of f, f' is weakly satisfied, too.
If O is a new object type, i. e. $O \notin V_f$, a path π from an object type $\overline{O} \in V_f$ to O exists, such that every node on this path, except $\overline{O}$, is a new type. More than one path of this kind may exist in $G_{\mathcal{S}}$. Since $G_{f'}$ is an OFD graph, one of these paths is selected, with $G_{f'}$ being the graph resulting from the combination of G_f and π. Following from this, α_{val_f} and $\alpha_{val_{f'}}$ differ in an attribute set consisting of the identifier attributes of the types of path π and, if $\varepsilon \subseteq attr(O)$, the attributes of ε. Besides the tuples from val_f extended to attribute set $\alpha_{val_{f'}}$, further tuples may exist in $val_{f'}$, especially tuples representing ε-objects/ε-values for which only partial entry combinations do exist (cf. Example 6.3). Therefore and because of the fact that ε is added to Δ and Γ, the weak satisfaction of f' follows from the weak satisfaction of f.

By Lemma 5.1, the augmentation rule is also applicable for f being strongly satisfied. The next example illustrates that in general the derived OFD f' is not strongly satisfied, even if f is strongly satisfied and ε refers to a node of G_f.

Example 6.3: Consider the schema from Example 6.2 and a state with the following extensions:

$$\begin{array}{ll} ext(O_1) = \{(1,[a])\} & ext(r_1) = \{(1,2)\} \\ ext(O_2) = \{(2,[b]),(3,[b'])\} & ext(r_2) = \{(2,4)\} \\ ext(O_3) = \{(4,[c])\} & \end{array}$$

OFD $g_1 : \{A\} \longrightarrow \{C\}$ is strongly satisfied, while $g_2 : \{A,B\} \longrightarrow \{C,B\}$ is not strongly but only weakly satisfied:

val_{g_1}

id_{O_1}	A	id_{O_2}	id_{O_3}	C
1	a	2	4	c

val_{g_2}

id_{O_1}	A	id_{O_2}	B	id_{O_3}	C
1	a	2	b	4	c
–	–	3	b'	–	–

Object $(3,[b'])$ of the new sink node O_2 of g_2 has no links. The null filter on attribute set $\{A,B\}$ discards the information about this object and surjectivity for O_2 and thus, strong satisfaction of g_2 is not given. Of course the same effect can be achieved by augmentation with a new node not yet included in the graph of the initial OFD. □

The rule covers the augmentation of a given OFD by a single object type or an attribute set of an object type. Obviously, any augmentation of an OFD by a set $\Phi \subseteq \mathcal{D}_S$ can be done by a sequence of elementary augmentations in the sense of the rule presented, as long as the graph resulting from adding the Φ-nodes (as well as connecting nodes and edges) is still an OFD graph. An "attribute set variant" of the augmentation rule is not needed since it could only be applied to attribute sets of the same object type, appearing in the left and right side of an OFD. Clearly, this is covered by the given rule.

Up to now, only rules yielding trivial dependencies or deriving a new OFD from a single dependency were considered. If an OFD is derived from two OFDs with graphs G_{f_1}, G_{f_2}, the subgraphs of G_{f_1} and G_{f_2} induced by the object types present in both OFDs have to match. This corresponds to the intuitive notion of combining the graphs via the subgraph common to both. Because of our restriction to cycle free OFD graphs cycles have to be avoided in the new graph. Both requirements are fulfilled if the intersection graph of G_{f_1} and G_{f_2} is connected and non empty. This is equivalent to the union graph of G_{f_1} and G_{f_2} being cycle free (cf. Lemma A.1 in the Appendix) and thus provides a simple precondition for the last three inference rules.

Union Rule

For a generalization of the union rule the effect described in Section 5.2 has to be taken into account. When two OFDs are combined, a constellation as shown in Figure 10 (c) becomes possible, for which the derived OFD is not satisfied. In principle, this may always be the case if two OFDs differ in at least one node, even for very simple OFD graphs:

for example, consider OFDs $g_1 : \{B,C\} \xrightarrow{G_{g_1}} \{D\}$ and $g_2 : \{B,C\} \xrightarrow{G_{g_2}} \{O_1\}$ of schema $\mathcal{S}$ from Figure 7. Obviously, the intersection graph of both OFDs, consisting of nodes O_2, O_3 and edge r_3, is connected. A state for $\mathcal{S}$ is easily constructed where g_1 and g_2 are (strongly) satisfied but the union OFD $\{B,C\} \to \{O_1,D\}$ of both is not. The same effect may occur with "non linear" OFDs of the schema from Figure 3 (b).

To avoid this problem, we restrict the union rule to OFDs with identical graphs:

> Let $f_i : \Delta \xrightarrow{G_{f_i}} \Gamma_i$, $i \in \{1,2\}$, be two OFDs with identical graphs.
> Then $f : \Delta \xrightarrow{G_{f_1}} \Gamma_1 \uplus \Gamma_2$ is strongly satisfied if f_1 and f_2 are strongly satisfied.

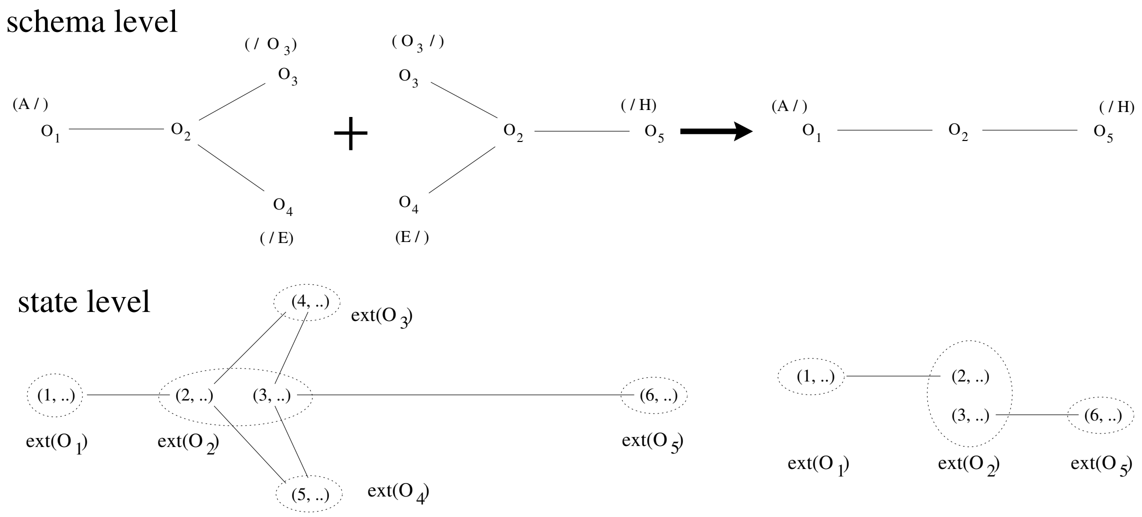


Figure 14: Violation of surjectivity by transitive combination of OFDs

According to our assumption, attributes and types are unique throughout the schema. Therefore the distribution of the Δ-labels is the same in G_{f_1} and G_{f_2}, and the $\mathcal{N}(\Delta)$-induced subgraphs of both graphs are identical. Labels of nodes that are sinks in both graphs are combined locally for each sink in the new OFD f. Obviously, no violation of strong satisfaction is caused by this. If nodes exist that are sinks of only one of the graphs G_{f_1} and G_{f_2}, they must be non leaf nodes or source nodes and it can be shown that the combination of both OFDs does not violate satisfaction, i. e. conflicts as in Figure 10 (c) cannot arise. Thus, f is strongly satisfied, if f_1 and f_2 are strongly satisfied.

Remark: A more general union rule can be given where the OFD graphs do not have to be identical. An additional dependency must be satisfied for the node of attachment of the intersection graph of both OFD graphs in G_{f_1} (G_{f_2}, respectively). For each of the two OFDs such a node may exist (e. g. in the previous example g_1 and g_2 have different nodes of attachment O_2 and O_3, whereas h_1 and h_2 have O_3 as common node of attachment). If Δ is an identification criterion for these nodes, the union OFD is strongly satisfied.

Transitivity Rule

In addition to cycles, surjectivity may cause problems in connection with transitivity of OFDs if strong satisfaction is considered. This can be illustrated by the example in Figure 14: if OFDs $g_1 : \{A\} \xrightarrow{G_{g1}} \{O_3, E\}$ and $g_2 : \{O_3, E\} \xrightarrow{G_{g2}} \{H\}$ are combined transitively in the usual way, this leads to OFD $g : \{A\} \xrightarrow{G_g} \{H\}$. With respect to the graphical representation of the extensions as given in Figure 14, it is easy to see that g_1 and g_2 are both strongly satisfied. The derived OFD g, however, is obviously not strongly satisfied under the same extensions. The reason for this is the lack of surjectivity for object type O_2, i. e. there are O_2-objects in the given extensions which are not reachable from any object of the source O_1. For the satisfaction of g_1 this has no consequences, because O_2 is no sink of g_1, but in the derived OFD g, O_2 becomes a "combining object type" on the path between source and sink, i. e. the graph of g is obtained by combining the graphs of g_1 and g_2 via this node.

A violation of surjectivity (and thus, of strong satisfaction) is always possible if the OFD graphs are connected by a node which was no sink in the first OFD. In the following, the notion of a combining object type will be clarified and based upon this a criterion for the avoidance of surjectivity violations will be given.

Definition 6.2: Let $f_1 : \Delta \xrightarrow{G_{f_1}} \Gamma$ and $f_2 : \Gamma' \xrightarrow{G_{f_2}} \Phi$, $\Gamma \subseteq \Gamma'$, be two OFDs, such that the union graph G_u of both graphs is cycle free. A node of attachment of G_{f_2} in G_u is called combining (object) type. □

Example 6.4: Object type O_2 is the only combining type of OFDs g_1 and g_2 from the example in Figure 14. □

A combining type of f_1 and f_2 exists if $V_{f_2} \subset V_{f_1}$, i. e. the graphs are not identical and G_{f_2} does not cover G_{f_1}. More than one combining type may exist for two OFDs. In the following we will focus on a transitivity rule for OFDs with exactly one combining object type.

If for the combining object type O of a transitively derived OFD graph the surjectivity property is given, no surjectivity violation can arise. Obviously, the sinks of an OFD satisfy this property. Therefore the OFD graphs should be combined via a sink type of the first OFD. However, it is easy to construct an example showing that this prerequisite alone is not sufficient yet. Surjectivity is needed at object level. Under the notion of strong satisfaction, this is not guaranteed if only attributes of a sink type appear in the OFD. Therefore O has not only to be a sink type, it must be an object type in the right side of the first OFD in order to get a sufficient condition.
To avoid cycles and to be sure that the two OFDs refer to the same subgraphs, identical trees induced by the intersection of the node sets are sufficient. Again this is given in the case of a cycle free union graph. It can be shown that this property follows from the existence of exactly one combining type.

> Let $f_1 : \Delta \xrightarrow{G_{f_1}} \Gamma$ and $f_2 : \Gamma \xrightarrow{G_{f_2}} \Phi$ be OFDs with $\Delta \cup \Phi \neq \emptyset$ and node labeling functions ν_{f_1}, ν_{f_2}, such that exactly one combining type O, $O \in \Gamma$, exists. Let $f : \Delta \xrightarrow{G_f} \Phi$ be the OFD with G_f being the $\mathcal{N}(\Delta \cup \Phi)$-tree of G and let the node-labeling function be given by
>
> $$\nu_f(O) := \begin{cases} (\delta, \emptyset) & \text{if } O \in \mathcal{N}(\Delta) \setminus \mathcal{N}(\Phi),\ \nu_{f_1}(O) = (\delta, \gamma) \\ (\emptyset, \gamma) & \text{if } O \in \mathcal{N}(\Phi) \setminus \mathcal{N}(\Delta),\ \nu_{f_2}(O) = (\delta, \gamma) \\ (\delta_1, \gamma_2) & \text{if } O \in \mathcal{N}(\Delta) \cap \mathcal{N}(\Phi), \nu_{f_1}(O) = (\delta_1, \gamma_1), \\ & \qquad \nu_{f_2}(O) = (\delta_2, \gamma_2) \\ \text{undefined} & \text{otherwise} \end{cases}$$
>
> Then f is strongly satisfied if f_1 and f_2 are strongly satisfied.

Note that the combining object type of f_1 and f_2 coincides with the node of attachment of the subgraph induced by G_{f_2} in the $\mathcal{N}(\Delta \cup \Phi)$-tree of the union graph.

The restriction of G_f to the $\mathcal{N}(\Delta \cup \Phi)$-tree of the union graph is necessary since G might contain Γ-nodes as leaves which are not needed in the new graph if they are no Δ- or Φ-nodes. Such nodes, including all nodes on the paths connecting them with the $\mathcal{N}(\Delta \cup \Phi)$-tree, can be omitted because they are not part of any path connecting $\mathcal{N}(\Delta)$- and $\mathcal{N}(\Phi)$-nodes. For example, the Γ-nodes O_3 and O_4 of the OFDs in Figure 14, including the edges connecting them to node O_2, are not needed in the derived graph.
Since $\mathcal{N}(\Gamma)$ is a subset of both node sets, the right side of f_1 and the left side of f_2 refer to the same subgraph. If any other object types common to both OFD graphs exist, they are connected in the same way in G_{f_1} as in G_{f_2}. Following from this and the surjectivity property, val_{f_2}

projected onto the Γ-attributes is a subset of val_{f_1} projected onto the same attributes, since every combination of $\mathcal{N}(\Gamma)$-objects in the given state is represented in val_{f_1}, except for combinations subsumed by others. As a consequence, for each combination t of values on the Φ-attributes in val_{f_2} a combination of Γ-attributes exists, determining t and also appearing in val_{f_1}. Γ-nodes may be deleted during the derivation of the new graph, but the prerequisite concerning the combining type guarantees that f_1- and f_2-linkages can be combined, i. e. a total entry combination exists for each Φ-combination in val_{f_2}. Thus, the Φ-attributes are determined by the Δ-attributes in val_f, too, if f_1 and f_2 are strongly satisfied. Therefore, f is strongly satisfied under the same state.

Remark: The transitivity rule (as well as the following pseudotransitivity rule) can be generalized to cover the combination of OFDs having no or more than one combining type. This will be addressed in forthcoming work.

Pseudotransitivity Rule

Pseudotransitivity can be handled similar to transitivity, with $f_2 : \Gamma \uplus \Theta \xrightarrow{G_{f_2}} \Phi$ and $f : \Delta \uplus \Theta \xrightarrow{G_f} \Phi$. Only the construction of the node-labeling function for f is slightly different since for nodes appearing in more than one of the sets $\mathcal{N}(\Delta)$, $\mathcal{N}(\Theta)$, and $\mathcal{N}(\Phi)$, the labels have to be combined by function ν_f, whereas labels corresponding to a set only appearing in Γ have to be omitted:

> Let $f_1 : \Delta \xrightarrow{G_{f_1}} \Gamma$ and $f_2 : \Gamma \uplus \Theta \xrightarrow{G_{f_2}} \Phi$ be OFDs, $\Delta \cup \Theta \cup \Phi \neq \varnothing$, with node labeling functions ν_{f_1}, ν_{f_2} such that exactly one combining type O, $O \in \Gamma$, exists. Let $f : \Delta \uplus \Theta \xrightarrow{G_f} \Phi$ be the OFD with graph G_f being the $\mathcal{N}(\Delta \cup \Theta \cup \Phi)$-tree of G and let the node-labeling function ν_f be defined by ν_f^1 and ν_f^2 as follows:
>
> $$\nu_f^1(O) := \begin{cases} \nu_{f_1}^1(O) & \text{if } O \in \mathcal{N}(\Delta) \setminus \mathcal{N}(\Theta) \\ \phi & \text{if } \phi \in \Theta,\ \phi \in sets^+(O),\ O \in \mathcal{N}(\Theta) \setminus \mathcal{N}(\Delta) \\ \nu_{f_1}^1(O) \uplus \phi & \text{if } \phi \in \Theta,\ \phi \in sets^+(O),\ O \in \mathcal{N}(\Delta) \cap \mathcal{N}(\Theta) \\ \varnothing & \text{if } O \in \mathcal{N}(\Phi) \setminus \mathcal{N}(\Delta \cup \Theta) \\ \text{undefined} & \text{otherwise} \end{cases}$$
>
> $$\nu_f^2(O) := \begin{cases} \nu_{f_2}^2(O) & \text{if } O \in \mathcal{N}(\Phi) \\ \varnothing & \text{if } O \in \mathcal{N}(\Delta \cup \Theta) \setminus \mathcal{N}(\Phi) \\ \text{undefined} & \text{otherwise} \end{cases}$$
>
> Then f is strongly satisfied if f_1 and f_2 are strongly satisfied.

Note that the use of the common union operation in the left side of f_2 would impose a restriction on the applicability of the rule.

The relational pseudotransitivity rule can be substituted by other relational rules. In general, this is not possible for the OFD rule set presented here if the notion of identification has to remain unchanged. The reason for this is that an augmentation of the left side of an OFD is only possible by using the augmentation rule. However, augmentation of a strongly satisfied non local OFD generally leads to a weakly satisfied OFD.

6.4 Additional inference rules

In addition to the relational inference rules adopted so far, further rules are needed for OFDs. First of all, these rules are concerned with transition from the object type level to the attribute level and vice versa.

Object Rule

The following simple rule represents the fact that an object is uniquely determined by its object identifier, i. e. the identifier always determines the object value. This allows to derive a set of local OFDs for each object type of the schema and can be regarded as a supplement of the reflexivity rule:

Let $O \in OT_S$ and $\gamma \in sets^+(O)$. Then $f : \{O\} \xrightarrow{G_f} \{\gamma\}$ with $G_f = (\{O\}, \emptyset, l_f)$ is strongly satisfied.

Lifting Rule

While the object rule allows a transition from object type level to attribute level, the next rule allows a transition to object type level, but only for source types. If a given OFD $f : \Delta \xrightarrow{G_f} \Gamma$ contains an attribute set $\delta \in sets^+(O)$ in the left side, δ can be replaced by the object type O: if f is strongly satisfied, a Δ-entry combination x leads to at most one Γ-combination y. If values are involved in x, more than one linkage may lead to y. Since different objects may have identical values, a replacement of values by objects results in a "refinement" of x: instead of δ, the identifier attribute id_O is now taken into account for checking strong satisfaction. Thus, the new OFD with δ substituted by O is strongly satisfied, too. The following rule formalizes this kind of change of source labels.

Let $f : \Delta \xrightarrow{G_f} \Gamma$ be a strongly satisfied OFD with $\delta \in \Delta$, $\delta \in sets^+(O)$ for some node O of G_f. Then $f' : (\Delta \setminus \{\delta\}) \cup \{O\} \xrightarrow{G_f} \Gamma$ is strongly satisfied, too.

Of course this only holds for a change of source labels. In general, a change of sink labels in the same manner leads to a violation of the OFD.

Shifting Rule

If an OFD is strongly satisfied by a state, the considered linkages guarantee unique access to sink objects or sink values. Using this, new OFDs can be derived by removing leaf nodes and modifying the source labels of such nodes in the following way:

Let $f : \Delta \xrightarrow{G_f} \Gamma$ with $|V_f| > 1$ be strongly satisfied with O being a source type of f that is a leaf of G_f and no sink, and node label $\nu_f(O) = (\delta, \emptyset)$. Assume O' to be the node connected to O in G_f via edge e. Then OFD $f' : \Delta' \xrightarrow{G_{f'}} \Gamma$ with

$$\begin{aligned}
&\Delta' := (\Delta \setminus \{\delta\}) \cup \{O'\},\\
&V_{f'} := V_f \setminus \{O\},\\
&E_{f'} := E_f \setminus \{e\}, \text{ and}\\
&l_{f'} = l_f|_{E_{f'}}
\end{aligned}$$

is strongly satisfied.

Note that due to the tree structure of OFD graphs, node O' and edge e' are uniquely determined.

Correctness of the shifting rule follows from simple observations concerning the linkages relevant for f: a Δ-total linkage l for f implies a Δ'-total linkage l' for f' by removing the O-object of l and its link belonging to e, and thus, provides a total entry combination for the same sink objects as l. From the entry combination of l, the entry combination of l' is constructed, where

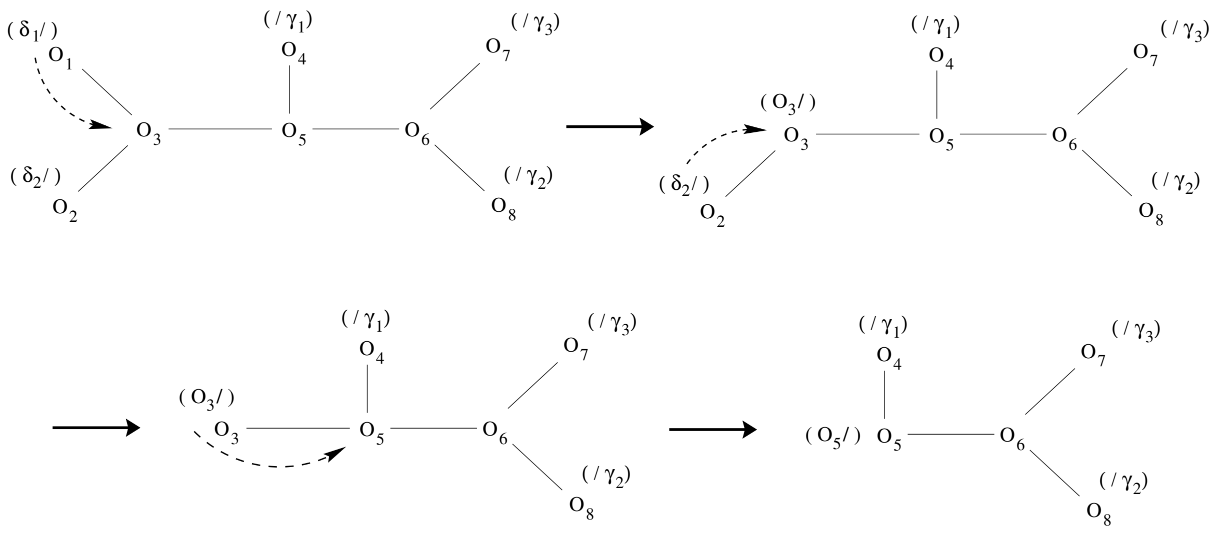
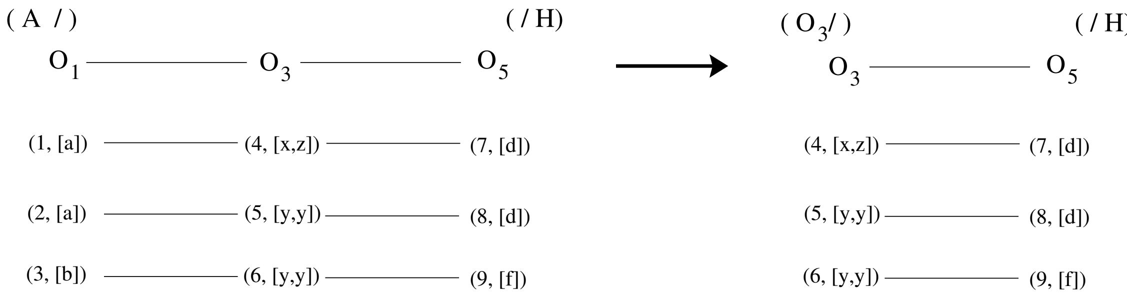


Figure 15: Application of the shifting rule

the entry value of δ in l is replaced by the O'-object of l. No collisions can arise from this (cf. lifting rule) and therefore f' is strongly satisfied if f is strongly satisfied.

The example in Figure 15 shows a sequence of applications of the shifting rule to an OFD. From OFD $\{\delta_1, \delta_2\} \xrightarrow{G} \{\gamma_1, \gamma_2, \gamma_3\}$ three new OFDs are derived by modifying the source labels δ_1, δ_2 and removing nodes O_1, O_2, and O_3. The OFD $g : \{O_5\} \xrightarrow{G_g} \{\gamma_1, \gamma_2, \gamma_3\}$ derived in the last step contains no source node which is also a leaf. Thus, the shifting rule is no more applicable to g. Moving non-leaf source labels within the OFD graph usually affects validity. It is easy to construct an example illustrating this.

Example 6.5: Consider OFD f_1 from Figure 16 and the state given below. The application of the shifting rule leads to OFD $f_1' : \{O_3\} \xrightarrow{G_{f_1'}} \{H\}$:



A shifting at attribute level from O_1 to O_3 is not possible for f_1: for every $\beta \in sets(O_3)$, $\{\beta\} \xrightarrow{G} \{H\}$ does not hold. □

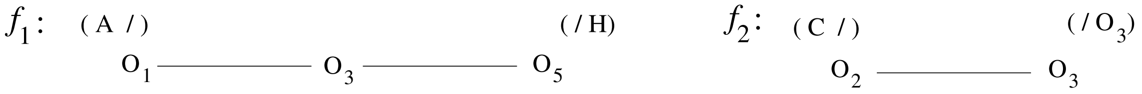


Figure 16: Necessity of the shifting rule

The necessity of the shifting rule is shown by OFDs f_1 and f_2 (Figure 16) of object schema from Figure 3 (b). By f_2 object type O_3 is determined which is neither source nor sink of f_1. The two OFDs cannot by combined by application of the transitivity rule. Using the previous rules except the shifting rule, an OFD g where attribute C of O_2 determines attribute H of O_5 cannot be derived. Obviously, OFD $f'_1 : \{O_3\} \xrightarrow{G_{f'_1}} \{H\}$ can be derived from f_1 by application of the shifting rule. Now g is derivable from f'_1 and f_2 by using transitivity. Thus, the shifting rule is necessary to derive new valid OFDs.

The shifting rule focuses on leaf nodes that are source but no sink types. Note that by application of the decomposition rule each source/sink-node can be transformed into a "pure" source node.

The shifting rule as defined above combines the transition from attribute to type level with source shifting, instead of allowing a shifting only for source labels that are object types. Otherwise, the application of the lifting rule would be necessary before using the shifting rule.

In summary we get the following theorem:

Theorem 6.1: Let $\mathcal{S}$ be an object schema with state $s(\mathcal{S})$. The inference rules (reflexivity, decomposition, augmentation, union, transitivity, pseudotransitivity, lifting, shifting, and object rule) for non local OFDs are sound. □

6.5 Cyclic OFDs

In this paper we focus on acyclic OFD graphs. However, especially in connection with transitivity, cycles may arise in graphs. These have to be avoided not only to meet the definition of an OFD graph: in such a case it cannot be expected that an OFD derived by transitivity from two valid OFDs is strongly satisfied by the same state. Consider the schema given in Figure 17. A customer may have one or more accounts at different branches of a bank, but

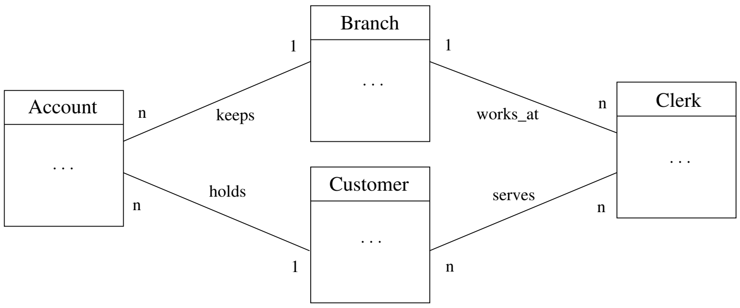


Figure 17: Example of a cyclic OFD graph

only one account per branch. Each customer has a clerk responsible for her/him at each branch where she/he holds an account, and each clerk is assigned to exactly one branch. Every account determines the branch keeping it and the customer holding it. A clerk is determined by a customer together with a branch; neither of the two object types alone suffices. Therefore the OFDs $f : \{Account\} \rightarrow \{Branch, Customer\}$ and $g : \{Branch, Customer\} \rightarrow \{Clerk\}$ do hold in all states. OFD $h : \{Account\} \rightarrow \{Clerk\}$ holds if for each *Branch/Customer* combination in val_g a matching combination in val_f exists. To check this, the construction of the validation relation by outer joins must not be linear. In the example above, a relation representing the *Account, Branch, Customer*-subgraph and one representing the *Branch,Customer,Clerk*-subgraph are needed. The validation relation of h has to be obtained by joining these two relations on the identifier attributes of *Branch* and *Customer*. Applying the usual linear construction of the validation relation, different combinations with the same *Customer*-object (or *Branch*-object) get mixed, falsifying the evaluation of h. What has to be ensured is that only matching *Customer*/*Branch*-combinations are combined during the construction of the validation relation. On that score, it is useful taking into account more than one path for a pair of object types involved in an OFD. This example shows that cyclic graphs may result from the combination of semantically meaningful OFDs and therefore should be considered, too.

7 Related work

7.1 Value based identification of objects

Many of the approaches to the specification of constraints at object schema level are concerned with specifying (value based) identification criteria or keys for object types, but not with dependencies for object schemas as a more general framework for this purpose. Related work, directed towards the definition and investigation of identification criteria, is discussed in this section.

Among others, Kim ([Kim95]) noted:

> "An OID does not carry any additional semantics. [...] It is more convenient for the user to be able to fetch one or more objects using user-defined keys."

Several approaches for the access to objects or entities have been discussed, especially in the context of other data models.

The restriction to a single relationship for the determination of a key for an entity type, similar to the weak entity concept of the Entity-Relationship Model ([Che76]), can be found in several approaches, e. g. [GMP88], [KZ95] or [RR94]. In [Zan79] for each record type of a network schema a set of *synonyms* (keys) is derived by inspecting different paths. Missing links are also considered: if an optional set type participates in a path, the result is a *pseudo synonym*. For the transformation to the relational model pseudo synonyms are ignored. In contrast to the approach presented in this paper, each synonym is determined using only one path in the schema.

In P/FDM ([PG88], [OE95]), a database system based on the functional data model, the key of a class C can be specified by using more than one path in the schema. A key is either a subset of the local attribute set of C or a set of relationships of C together with a local attribute subset. Along each relationship the whole key of the related class is used to build a key for C by expanding the key specifications recursively. The use of non-key attributes from other classes as well as the construction of a key exclusively using foreign keys is not possible. Furthermore, each class must have a key in the aforementioned sense and only single valued relationships are used in the process.

Schewe and Thalheim ([ST93]) present a formal approach to VBICs in the context of an object oriented data model. In addition to using only the type of a class itself for the determination of VBICs, leading to the notions of *value identifiability* and *value representability*, non local identification criteria are also considered: a class C_1 is called *weak value identifiable* if a value

identifiable class C_2 and a path of relationships from C_2 to C_1 exists, where each participating relationship, regarded as a directed reference from C_2 to C_1, has to be a surjective function. This corresponds to an OFD $f : \{C_2\} \xrightarrow{G_f} \{C_1\}$ with G_f being a chain. Only one object type is used and no combination with local attributes is allowed. As in [Zan79] a single path is considered. Furthermore, the requirement of every relationship of the sequence to be surjective is not necessary and does not allow the use of partial linkages. Surjectivity is too restrictive, even if no partial linkages or insufficient link chains of C_1-objects appear in a given state: obviously, f might hold even if not every object of the classes participating in a chain is involved in a sequence of links leading to a C_1-object. Considering only surjective relationships would be an unnecessary restriction.
Some of the results from [ST93] can also be found in [Kos95] where, in addition to the object identifier and the complete object value, key specifications for schemas are considered to distinguish objects. Objects (or their values) of other classes are taken into account by so-called *external keys*, without considering how to obtain these keys.

The possible use of the existence of values for the distinction of objects is mentioned in [BT95], where the use of tree structures for the identification of entities in an extended ER model is sketched also. Entity identification is reduced to identification in nested relations where different types of equality are considered. Axiomatization or inference rules are not given.

In the data model proposed for database design by Rosenthal and Reiner ([RR94]), primary keys are allowed to include null attributes as long as unique identification of entities is guaranteed. No further remarks on the type of identification implied by this approach are given. A *key set* $\mathcal{K}$ as introduced in [Tha89] allows the specification of keys with null attributes. However, they are not used for accessing tuples in a relation since for each tuple t there has to exist at least one key κ in $\mathcal{K}$ such that t is total on κ.

In [AVdB95] the use of the complete object value for distinguishing objects in the sense of deep equality is investigated. An object value is unfolded by dereferencing object identifiers appearing in it and value based identification is defined with respect to this unfolded value. This corresponds to a programming language view of the environment of objects in the database by using directed references with no possibility to follow pointers backwards. Relationships in an OMT-like object schema are not directed (although they can be implemented by inverse directed references, of course). As a consequence, the environment of an object under the database view subsumes the environment as seen under the programming language view. Therefore it is possible that two deep equal objects are distinguishable if their complete relationship environment in the database is taken into account.

7.2 Generalized functional dependencies

In [Lee95], *object functional dependencies* are introduced as FDs on the attribute set of a single class. Among these attributes, complex attributes, i. e. attributes of collection type or attributes that are references to other classes, are allowed. Reference attributes represent relationships and thus it is possible to specify an object functional dependency $\beta \rightarrow R$, where β is an attribute set determining the reference attribute R. This restricts the relationship represented by R to a function and leads to a notion of identification similar to weak entities, with the class referenced by R corresponding to the weak entity type. However, this is the only way to specify a dependency involving more than one class. Apart from this case, object functional dependencies as proposed by [Lee95] refer to one class only. More general non local dependencies cannot be expressed.

Weddell ([Wed90], [Wed92]) introduces *path functional dependencies* as a generalization of FDs for semantic data models. This generalization is based upon a restricted data model: a class schema is given by a set of complex object types, with each type defined by a set of attributes, called properties. A property is either of atomic type or of object type. In the latter case, the property represents a directed relationship between the two types. Properties and directed paths of properties have to be functions and they are mandatory. As a consequence an object o of type O must have exactly one link for each relationship that O participates in. Relationships of more general cardinality have to be modeled by introducing additional linking types, as known from the network data model. For a class schema, a path functional dependency, denoted as $C(X \to Y)$, is defined by specifying two sets X and Y of property paths with respect to a single type C, the "center type". All paths of X and Y originate from this type. The constraint is satisfied, if for each pair of C-objects with identical values on the properties given by X, the property values on Y coincide, too. Relationships directed towards C are not taken into account. By definition, surjectivity is only guaranteed for the center type. Thus, only type canonical constraints can be expressed and the specification of an identification criterion[4] for a type C' by a path functional dependency p requires C' to be the center type of p.
Inference rules for path functional dependencies are presented by generalizing relational rules, as well as by adding rules which allow to modify the paths involved in a dependency.
However, even for simple object schemas (e. g. the schema from Figure 1 (a)) it is difficult to obtain a class schema representing the same semantics as the object schema. For example, class schemas offer no inverse properties for representing undirected, bidirectional relationships. The use of additional linking types is in conflict with the requirement of properties to be mandatory.

7.3 The validation relation

A validation relation is constructed by joining the relational representations of types and relationships in appropriate order using identifier attributes. This operational proceeding is equivalent to navigating through the database along links between objects and is closely related to the notion of *maximal traversals*, maximal sets of connected records obtained by navigation in a network database, as introduced by Lien ([Lie82]). Each maximal traversal corresponds to a partial tuple and in the case of a cycle free network schema a partial relation is obtained, representing the database state. This coincides with the construction of the unnormalized validation relation $\mathcal{V}$ where the underlying OFD graph is regarded as a network schema and link chains as maximal traversals. Following from this, no tuple of $\mathcal{V}$ is subsumed by any other tuple in $\mathcal{V}$. Concerning identification, the rs- and sub-normalization steps are applied to $\mathcal{V}$ additionally. Furthermore, only those object types and attributes of a schema which are necessary for the evaluation of an OFD have to be considered in the validation relation. In general, this is not the whole schema.

Lee and Wiederhold ([LW94]) describe the construction of objects from sets of tuples, starting with a *pivot relation* and using the *left outer join* operation. Their main emphasis is on the construction process, not on determining VBICs. The identifier of an object is provided simply by

[4] Using a notion of identification like strong satisfaction where each object is to be identified in order to ensure surjectivity.

the key of the pivot relation; it could be used as a (local) VBIC since it is no object identificator in the sense of an internal identifier. Nevertheless, building objects from relations is related to building a validation relation val_f. In the case of type canonical OFDs the method can be applied: the construction process must start with the sink object type O of f and the sequence of operations has to be chosen appropriately. This is necessary to ensure that every O-object is represented in val_f. Also the left outer join would eliminate the need for the rs-normalization step in this case, which in terms of [LW94] is simply the application of a *non-null filter*. Equivalently, it can be seen as a restriction to traversals ([Lie82]) total on the attributes belonging to the right side object type. However, sub-normalization is still necessary when using the left outer join. The aforementioned generalization of the surjectivity requirement from [ST93] is not affected by changing the join operator. Given non type canonical OFDs, the approach of Lee and Wiederhold is not applicable for our purposes. In this case the FOJ has to be used, since no unique pivot relation is given to start the construction.

7.4 Relational theory

The construction of the validation relation of a non local OFD can be formulated using some well known terms of relational theory, too. It is easy to see that a join tree ([BFMY83]) is obtained by the following modifications to an OFD graph G_f: insertion of relationship nodes, adjustment of edge labels and nodes. Due to the construction, only identifier attributes corresponding to a single object type appear as edge labels. Because of the monotonicity of the FOJ operation, results from relational theory concerning the natural join operation are applicable in this case, too. The relational database schema implied by G_f has the running intersection property ([BFMY83]). From the join tree of G_f a monotone, sequential join expression π can be derived ([BFMY83]). val_f (unnormalized) is obtained by applying π to the relational representations.

Viewing the OFD graph itself as a database schema, the validation relation is also related to the basic idea of the universal relation approach ([Mai83], [Ull88]). There is a correspondence between the universal relation and the construction of a validation relation in the case of an acyclic relational database schema.

8 Outlook

This paper presents a framework for the specification of constraints between attributes and object types in the context of a simple object oriented data model by using OFDs. With this kind of dependency, (value based) identification criteria can be defined directly or derived from a given set of OFDs by applying inference rules. The derivation process for this has to be investigated more thoroughly. Especially with respect to the determination of VBICs, the schema structure has to be taken into account. It is necessary to translate schema elements with relevance for identification, like relationships of restricted cardinality or inheritance hierarchies, into OFDs. They have to be added to the set of dependencies that are specified explicitly for an object schema. For this, the simplified object model we considered in this paper has to be extended to include inheritance.

In general, it cannot be expected that for each object type of a schema a "natural" VBIC is specified or derivable. Thus, in some cases artificial identifier attributes have to be introduced in order to supply a VBIC for a type. This should be done under consideration of the existing identification dependencies, i. e. the introduction of an identifier attribute for a single type should provide VBICs for other types as well, if possible. Criteria for the placement of such attributes in a schema are needed, using the information given by the OFD set of the schema.

The notion of OFDs considered in this paper allows to specify conjunctive, possibly non local VBICs. However, some VBICs, e. g. those from the examples in Figures 1 (c) and 2, can only be "simulated" by choice of an appropriate notion of identification. For these VBICs a further notion of identification is needed where missing links may be present but are not exploited for identification purposes. This leads to disjunctive VBICs or set oriented VBICs.

The views of identification discussed in this paper arise naturally if the OFD approach is taken. The schema state can also be represented by a graph consisting of objects as nodes and links as edges. Identification criteria can be defined with respect to this graph by considering the link environment of an object and defining subgraphs "centered" around the object. Different ways to distinguish objects at this level should be investigated and compared to identification as proposed by the OFD approach.

A Graphtheoretic notions

Definition A.1: For a finite set V let $\mathcal{P}_2(V)$ denote the set of all subsets of V with cardinality 2. Let $G=(V,E)$ be a connected finite graph with node set V and edge set $E \subseteq \mathcal{P}_2(V)$. G is a <u>tree</u> if G is cycle free. A <u>subgraph</u> of G is a graph $\overline{G}=(\overline{V},\overline{E})$ with $\overline{V} \subseteq V$ and $\overline{E} \subseteq (E \cap \mathcal{P}_2(\overline{V}))$. A <u>node of attachment</u> of subgraph $\overline{G}$ in graph G is a node of $\overline{G}$ that is incident in G with an edge not belonging to $\overline{E}$. Let $V' \subseteq V$ be a non empty subset of V. A <u>V'-covering subgraph</u> of G is a connected subgraph $\overline{G}=(\overline{V},\overline{E})$ of G with $V' \subseteq \overline{V}$. A <u>$V'$-tree</u> of G is a minimal V'-covering subgraph $\overline{G}=(\overline{V},\overline{E})$ (i. e. there exists no V'-covering subgraph $\overline{G'}=(\overline{V'},\overline{E'})$ with $\overline{V'} \subset \overline{V}$), such that $\overline{G}$ is a tree. The <u>intersection graph</u> G of two graphs $G_1=(V_1,E_1)$, $G_2=(V_2,E_2)$ is defined as $G := (V_1 \cap V_2, (E_1 \cap E_2)|_{\mathcal{P}_2(V_1 \cap V_2)})$. If G_1, G_2 are edge-labeled graphs, the labels of edges in $E_1 \cap E_2$ have to match, too. The <u>union graph</u> G of G_1 and G_2 is defined as $G := (V_1 \cup V_2, E_1 \cup E_2)$. For edge-labeled graphs the edge labeling function of G is derived from the labels of G_1 and G_2. □

The definition of a V'-tree for edge-labeled graphs is analogous. Note that for a tree $G=(V,E)$ each V'-tree of G is unique.

Definition A.2: Let $G=(V,E)$ be a connected graph and $V' \subseteq V$. Let $G_1=(V_1,E_1)$, $G_2=(V_2,E_2)$ be two V'-trees of G. G_1 and G_2 are <u>identical</u> iff $V_1=V_2$ and $E_1=E_2$. In case of G_1 and G_2 being edge-labeled graphs, the labels have to match, too. □

Lemma A.1: Let trees $T_1=(V_1,E_1)$ and $T_2=(V_2,E_2)$ be subgraphs of a connected graph $G=(V,E)$ with $V_1 \cap V_2 \neq \varnothing$. The following conditions are equivalent:

(i) The union graph of T_1 and T_2 is cycle free (i. e. is a tree).

(ii) The intersection graph of T_1 and T_2 is connected.

(iii) The $(V_1 \cap V_2)$-trees of T_1 and T_2 are identical.

□

References

[AH84] S. Abiteboul and R. Hull. IFO: A formal semantic database model. In *Proceedings of the 3rd ACM Symposium on Principles of Database Systems (PoDS), Waterloo*, pages 119–132, 1984.

[AK89] S. Abiteboul and P. C. Kanellakis. Object identity as a query language primitive. In *Proceedings ACM SIGMOD Conference on Management of Data, ACM SIGMOD Record, 18(2)*, pages 159–173, 1989.

[AVdB95] S. Abiteboul and J. Van den Bussche. Deep equality revisited. In *Proceedings 4th Conference on Deductive and Object-Oriented Databases, Singapore*, volume 1013 of *Lecture Notes in Computer Science*, pages 213–228. Springer-Verlag, 1995.

[Bee93] C. Beeri. Some thoughts on the future evolution of object-oriented database concepts. In W. Stucky and A. Oberweis, editors, *Datenbanksysteme in Büro, Technik und Wissenschaft,* 5. GI-Fachtagung, Braunschweig, pages 18–32. Springer-Verlag, 1993.

[BFMY83] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.

[BT95] C. Beeri and B. Thalheim. Identification as a primitive of database models. preliminary version, 1995.

[Che76] P. P. S. Chen. The entity – relationship model – toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[CK86] G. P. Copeland and S. N. Khoshafian. Object identity. In *Proceedings OOPSLA 1986, Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, ACM SIGPLAN, Vol. 1(11)*, pages 159–173, 1986.

[Cod79] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[DV93] K. Denninghoff and V. Vianu. Database method schemas and object creation. In *Proceedings of the 12th ACM Symposium on Principles of Database Systems (PoDS), Washington*, pages 265–275, 1993.

[GMP88] P. M. D. Gray, D. S. Moffat, and N. W. Paton. A Prolog interface to a functional data model database. In J. W. Schmidt, S. Ceri, and M. Missikoff, editors, *Proceedings Advances in Database Technology - EDBT '88, Venice*, volume 303 of *Lecture Notes in Computer Science*, pages 34–48. Springer-Verlag, 1988.

[Gog95] M. Gogolla. A declarative query approach to object identification. In *Proceedings 14th International Conference on Object-Oriented and Entity-Relationship Modeling, Gold Coast*, volume 1021 of *Lecture Notes in Computer Science*, pages 65–76. Springer-Verlag, 1995.

[HOT76] P. Hall, J. Owlett, and S. Todd. Relations and entities. In G.M. Nijssen, editor, *Modelling in Data Base Management Systems*, pages 201–220. North-Holland, 1976.

[HY91] R. Hull and M. Yoshikawa. On the equivalence of database restructurings involving object identifiers. In *Proceedings of the 10th ACM Symposium on Principles of Database Systems (PoDS), Denver*, pages 328–340, 1991.

[JQ92] H.V. Jagadish and X. Qian. Integrity maintenance in an object-oriented database. In *Proceedings 18th Conference on Very Large Database Systems, Vancouver, Canada*, pages 469–480. VLDB Endowment, 1992.

[Kim95] W. Kim. Object-oriented database systems: Promises, reality, and future. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability and Beyond*, chapter 13. ACM Press, 1995.

[Kos95] A. S. Kosky. Observational distinguishability of databases with object identity. Technical report, Department of Computer and Information Science, MS-CIS-95-20, University of Pennsylvania, 1995. also in: P. Atzemi, V. Tannen (eds.): Proceedings of the Fifth International Workshop on Database Programming Languages (DBPL-5), Gubbio, Italy, Springer-Verlag, 1995.

[KR97] H. J. Klein and J. Rasch. Value based identification and functional dependencies for object databases. In *Data Management Systems - Proceedings of the Third International Basque Workshop on Information Technology (BIWIT 97), Biarritz, France*, pages 22–32. IEEE Computer Society Press, 1997.

[KZ95] M. Kolp and E. Zimanyi. Relational database design using an ER approach and Prolog. In S. Bhalla, editor, *Proceedings 6^{th} Conference on Information Systems and Management of Data (CISMOD), Bombay, India*, volume 1006 of *Lecture Notes in Computer Science*, pages 214–231. Springer-Verlag, 1995.

[Lee95] B. S. Lee. Normalization in OODB design. *ACM SIGMOD Record*, 24(3):23–27, 1995.

[Lie82] Y. E. Lien. On the equivalence of database models. *Journal of the ACM*, 29(2):333–362, 1982.

[LP76] M. Lacroix and A. Pirotte. Generalized joins. *ACM SIGMOD Record*, 8(3):15–16, 1976.

[LW94] B. S. Lee and G. Wiederhold. Outer joins and filters for instantiating objects from relational databases through views. *IEEE Transactions on Knowledge & Data Engineering*, 6(1):108–119, 1994.

[Mai83] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[MUV84] D. Maier, J. D. Ullman, and M. Y. Vardi. On the foundations of the universal relation model. *ACM Transactions on Database Systems*, 9(2):283–308, 1984.

[MW90] T. Matsushima and G. Wiederhold. A model of object-identities and values. Technical report, Stanford University, USA, 1990. Report STAN-CS-90-1304.

[OE95] Object Database Group and S. Embury. User manual for P/FDM V.10.1. Technical report, Department of Computer Science, University of Aberdeen, U.K., AUCS/TR9501, 1995.

[PG88] N. W. Paton and P. M.D. Gray. Identification of database objects by key. In K. R. Dittrich, editor, *Advances in Object-Oriented Database Systems, Proceedings 2. International Workshop on Object-Oriented Database Systems, Bad Münster, Germany, 1988*, volume 334 of *Lecture Notes in Computer Science*, pages 280–285. Springer-Verlag, 1988.

[Rat97] Rational Software Corporation, Santa Clara, USA. *Unified Modeling Language (UML) Notation Guide, Version 1.0*, 1997.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

[RR94] A. Rosenthal and D. Reiner. Tools and transformations-rigorous and otherwise-for practical database design. *ACM Transactions on Database Systems*, 19(2):167–211, 1994.

[ST93] K. D. Schewe and B. Thalheim. Fundamental concepts of object oriented databases. *Acta Cybernetica*, 11(1–2):49–83, 1993.

[Tha89] B. Thalheim. On semantic issues connected with keys in relational databases permitting null values. *Journal Information Processing and Cybernetics EIK*, 25(1/2):11–20, 1989.

[Ull88] J. D. Ullmann. *Principles of Database and Knowledge-Base Systems*, volume I and II. Computer Science Press, 1988.

[WdJ91] R. Wieringa and W. de Jonge. The identification of objects and roles – object identifiers revisited. Technical report, Vrije Universiteit Amsterdam, 1991. Report IR–267, Faculteit de Wiskunde en Informatica Vrije Universiteit Amsterdam.

[Wed90] G. E. Weddell. A theory of functional dependencies for object-oriented data models. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Proceedings of the First International Conference on Deductive and Object-Oriented databases (DOOD89), Kyoto, Japan, 1989*, pages 165–184. Elsevier Science Publisher B.V. (North-Holland), 1990.

[Wed92] G. E. Weddell. Reasoning about functional dependencies generalized for semantic data models. *ACM Transactions on Database Systems*, 17(1):32–64, 1992.

[Zan79] C. Zaniolo. Design of relational views over network schemas. In *Proceedings ACM SIGMOD International Conference on Management of Data, Boston*, pages 179–190, 1979.