VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematical and Computing Sciences
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

Problems with OOFNF: a proposed normal form for object oriented databases

Gillian Dobbie and Peter Andreae

Technical Report CS-TR-99/8
June 1999

Abstract

Update anomolies arise in relational databases if the database schema is poorly designed. Normal forms that have been defined over relational database schemas disallow redundant data in the resulting database. Without redundancy, update anomolies don't occur. Update anomalies can also arise in object oriented databases if the schema is poorly designed. Several definitions of normal forms over object oriented databases have been proposed.

We are particularly interested in object oriented functional normal form (OOFNF). While the intuition behind OOFNF is good, the details of the definition are wrong. For example, the definition of OOFNF is based on the concept of path functional dependencies (PFD) that have been defined over single valued properties. The authors of the paper defining OOFNF neither extend the definition of PFDs to multi-valued properties, nor do they restrict their own data model to single valued prpoperties.

In this paper, we redefine OOFNF over a restricted data model with single valued properties.

Author Information

Gill and Pondy can be contacted by email:{gill,pondy}@mcs.vuw.ac.nz. This work follows Gill's interest. Pondy reluctantly worked in the area initially until he discovered that there were some interesting problems here.

1 Introduction

Because update anomalies can occur in poorly designed object oriented databases, it is useful to define a concept like normalization to help designers build object oriented database schemas [2, 4]. Several definitions of normal forms over object oriented databases have been proposed.

One approach is described by Li et al. [5]. They define object oriented functional normal form (OOFNF) which is based on path functional dependencies (PFDs) [9]. They claim that their definition is equivalent to Boyce-Codd Normal Form (BCNF) if the classes contain only built-in typed properties.

While the intuition behind OOFNF is good, the details are wrong. The definition of object oriented functional normal form (OOFNF) is based on Weddell's definition of path functional dependencies [9]. Path functional dependencies are defined on a data model that has single valued properties. Li et al. claim to define OOFNF over databases that have collection properties but use Weddell's axioms without extension, and without limiting their data model to the model used by Weddell.

Another flaw is that the definition of OOFNF is not equivalent to BCNF as claimed in their paper, even when restricted to single valued properties, with only built-in typed properties.

In this paper we address these problems. We start by introducing terminology that is needed later in this paper. In section 3 we redefine OOFNF on a restricted data model with only single valued properties and prove that the new definition is equivalent to BCNF when restricted to built-in types.

2 Background

Before we discuss the details of the work of Li et al, we introduce terminology that is needed later in this paper.

Example 2.1 Consider the following schema for an OO database.
$order\{order\_number : string,\ \ order\_lines : \{orderline\}\}$
$orderline\{item : string,\ \ unit : string,\ \ quantity : integer\}$ $\triangle$

This is a schema with **classes** *order* and *orderline*. The **properties** of *order* are *order_number* and *order_lines*. The property *order_number* is a **single valued property** and *order_lines* is a **multi-valued** or **collection property**.

A **path function** is a sequence of properties that describes a path through classes in a schema. Consider the path function $P_1.P_2.P_3$ and suppose P_i is a property of C_i. The type of each property P_i (except the last) is class C_{i+1}. The domain of a path function is the class containing the first property in the path function (i.e. C_1 in this example), and the range of a path function is the type of the last property. For example, *order_lines.quantity* is a path function with domain *order* and range *integer*. Each class has an **identity path function** denoted as *Id*. The identity path function is a way of referring to property value paths of length zero. **PathFuncs(C)** is the set of path functions with domain C.

Path functional dependencies (PFDs) express constraints between sets of path functions, similar to functional dependencies in the relational model. For example, the path functional dependency $\{order\_number\} \rightarrow \{order\_lines\}$ indicates that no two objects in class *order* will have different values for *order_lines* if they have the same value for *order_number*. A functional dependency is said to be **trivial** if the righthand side is contained in the lefthand side of the functional dependency. Similarly functional dependencies are **left-reducible (right-reducible)** if the lefthand (righthand) side of the dependency can be reduced without losing any information. These terms are also applicable to PFDs, and a PFD $X \to Y$ is said to be **reduced** if it is nontrivial, left-reduced and right-reduced. If D is a set of PFDs, then we use D^+ to refer to the closure of D [9].

An instance of the schema introduced in example 2.1 follows.

order_number	order_lines		
	item_name	unit	quantity
01	PC	set	1
	Cable	m	10
	Floppy	box	20
02	PC	set	1
	CD-ROM	pce	5

The class *orderline* is a **dependent class** because no property uniquely identifies each object in the class. The class *order* is an **independent class**, because the property *order_number* uniquely identifies each object in the class. The property *item_name* is a **partition key**, as it identifies *orderline*s within *order*s. **Semantic keys** or **SKeys** are like keys in the relational model. Independent classes have SKeys. Dependent classes do not have SKeys.

Li et al. also introduce **location keys** or **LKeys**. The intuition behind LKeys is that location keys in $LKeys(C, P)$ contain the semantic and partition keys of the classes that are traversed by the path function P starting from class C.
For example, $LKeys(order, order\_lines.unit) = \{\{order\_number, item\_name\}\}$. The definition in [5] does not quite capture this intuition.

3 OOFNF with single valued properties

In this section we work with a data model that is restricted to single valued properties. We demonstrate that some definitions in [5] are wrong and correct them.

We start this section with definitions of object oriented functional normal form (OOFNF) and Boyce Codd Normal Form (BCNF) and demonstrate that the definition of OOFNF is not equivalent to BCNF when restricted to an example of an independent class, with single valued properties, and with only built-in typed properties. We redefine location keys and extend the definition of reduced PFDs, showing that these definitions work with various examples. Finally we prove that with the new definitions there is an equivalence between OOFNF and BCNF.

Definition (from [5]) Given an OO schema S, an independent class $C \in Classes(S)$ and the set D of PFDs over S, class C is said to be in **OO functional normal form** (OOFNF) *iff* for every reduced path functional dependency $X \to P \in D^+$ where $X \subseteq PathFuncs(C)$ and $P \in PathFuncs(C)$, there exists $K \in LKeys(C, P)$ such that $X \to K$ and $K \to P$ both hold.
Schema S is in OOFNF if each independent class in S is in OOFNF.

The intuition behind the definition of OOFNF is roughly that every property that you can get to from a class must be dependent on the key of that class and any location keys along the way.

The definition of BCNF is naturally simpler because the data model that it is defined over is less complex, but the intuition is similar — each property in the relation must be dependent on the key of that relation.

Definition A relation is in **Boyce Codd Normal Form** (BCNF) if the only non-trivial dependencies are those in which a key functionally determines one or more other attributes. A schema is in BCNF if each relation in the schema is in BCNF.

In [5], Li et al claim that their definition of OOFNF is equivalent to BCNF if the classes contain only built-in typed properties. In the following example we demonstrate that this claim is not correct.

Example 3.1 Consider the following schema:

$supplier\{name : string,\ \ item : string,\ \ price : string\}$,

with path functional dependencies $D = \{\{name, item\} \rightarrow Id,\ \ \{name, item\} \rightarrow price\}$. According to the definition of location key (LKeys) in Li et al. [5], there is only one location key for this class namely $LKeys(supplier, Id) = \{\{name, item\}\}$. Now $\{name, item\} \rightarrow price$, so according to the definition of OOFNF we require a definition for $LKeys(supplier, price)$. As this is undefined, it follows that *supplier* is not in OOFNF.

Consider now the corresponding relational schema:

$supplier\{name : string,\ \ item : string,\ \ price : string\}$,

with key $\{name, item\}$ and functional dependencies $\{\{name, item\} \rightarrow price\}$. This relation is in BCNF. $\triangle$

Using the definitions in [5], we do not get an equivalence between OOFNF and BCNF. The intuition behind LKeys suggests that the LKeys for the *supplier* scheme should be the same as the key in the relation database, in which case we would get the equivalence we expect. In fact, if all properties are single-valued, then there are no partition keys, so that the following redefinition of LKeys will give the desired equivalence between OOFNF and BCNF.

Definition Let C be a class definition in schema S, where C is an independent class. Let $pf \in PathFuncs(C)$. The set of *location keys* of C with respect to pf is defined as follows:

$LKeys(C, pf) = SKeys(C)$.

Example 3.2 Consider the schema and path functional dependencies in example 3.1. The location keys are now $LKeys(supplier, Id) = \{\{name, item\}\}$, $LKeys(supplier, name) = \{\{name, item\}\}$, $LKeys(supplier, item) = \{\{name, item\}\}$, $LKeys(supplier, price) = \{\{name, item\}\}$. Now for each of the reduced functional dependencies $X \rightarrow P \in D^+$, namely $\{name, item\} \rightarrow Id$ and $\{name, item\} \rightarrow price$, there is an LKey $K = \{name, item\}$ such that $X \rightarrow K$ and $K \rightarrow P$, so the class *supplier* is in OOFNF. $\triangle$

We have shown that if we restrict ourselves to a data model with only single valued properties and use the new definition of LKeys, we get the expected result for the schema in example 3.1. However, there is a further problem with the definitions that arises in more complex schemas. The next example contains a property whose type is another class.

Example 3.3 Consider the following OO schema:

$order\{s : supplier,\ \ item : string,\ \ price : string\}$,

$supplier\{name : string,\ \ address : string\}$,

with path functional dependencies $D = \{\{s, item\} \rightarrow Id,\ \ \{s, item\} \rightarrow price,\ \ name \rightarrow Id,\ \ s.name \rightarrow s.address,\ \ name \rightarrow address\}$.

Using our new definition, the LKeys are

$LKeys(order, P) = \{\{s, item\}\}$, for all $P \in PathFuncs(order)$,

$LKeys(supplier, P) = \{\{name\}\}$, for all $P \in PathFuncs(supplier)$. $\triangle$

All but one of the PFDs satisfy the OOFNF constraint (using the new definition of LKeys), but the PFD $s.name \rightarrow s.address$ is problematic. The PFD $s.name \rightarrow s.address$ is reduced (nontrivial, left reduced, and right reduced), but it does not satisfy the OOFNF constraint. The $LKeys(order, s.address) = \{\{s, item\}\}$ but it is not the case that $s.name \rightarrow$

$\{s, item\}$ and $\{s, item\} \rightarrow s.address$. Since there is no K in $LKeys(order, s.address)$ such that $s.name \rightarrow K$ and $K \rightarrow s.address$, the *order* class is not in OOFNF although we would expect it to be.

The problem is that the PFD $s.name \rightarrow s.address$ is being tested with respect to the *order* class, instead of the *supplier* class, even though the range of every path function in the PFD is a property in the *supplier* class. We can require that each functional dependency is tested in the class that it is most applicable to by introducing an additional clause into the definition of reduced PFDs that removes prefixes that are common to all the path functions in a PFD.

Definition Given a set D of PFDs over schema S, a PFD $X \rightarrow Y$ in D^+ is said to be

1. *trivial* if $Y = \emptyset$, or $\forall pf \in Y$, $\exists pf' \in X$ such that $\exists pf'' \in PF(S)$, and $pf = pf' \circ pf''$
2. *left-reducible* if $\exists X', X' \subset X$ such that $X' \rightarrow Y$ is in D^+.
3. *right-reducible* if $\exists Y', Y' \subset Y$ such that $Y' \rightarrow Y$ is a nontrivial PFD in D^+.
4. *prefix reducible* if $\exists pf' \in PF(S)$ such that for each $pf \in \{X \cup Y\}$, $\exists pf'' \in PF(S)$ such that $pf = pf' \circ pf''$

A PFD $X \rightarrow Y$ is said to be *reduced* if it is nontrivial, left-reduced, right-reduced and prefix-reduced.

Reconsidering example 3.3, the functional dependency $s.name \rightarrow s.address$ is no longer a reduced functional dependency. All the reduced functional dependencies satisfy the OOFNF constraint so this schema is in OOFNF as expected.

The next example shows that the new definitions of LKeys and reduced PFD's work on a schema with more than one level of composition.

Example 3.4 Consider the following schema:
$customer\{cname : string,\ \ caddress : string,\ \ corder : order\}$,
$order\{s : supplier,\ \ item : string,\ \ price : string\}$,
$supplier\{name : string,\ \ address : string\}$,
with reduced path functional dependencies $D = \{\{cname\} \rightarrow Id,\ \ \{cname\} \rightarrow caddress,$ $\{cname\} \rightarrow corder,\ \ \{s, item\} \rightarrow Id,\ \ \{s, item\} \rightarrow price,\ \ name \rightarrow Id,\ \ name \rightarrow address\}$.
The location keys are
$LKeys(customer, P) = \{\{cname\}\}$, for all $P \in PathFuncs(customer)$,
$LKeys(order, P) = \{\{s, item\}\}$, for all $P \in PathFuncs(order)$,
$LKeys(supplier, P) = \{\{name\}\}$, for all $P \in PathFuncs(supplier)$.

Now for each reduced functional dependency $X \rightarrow P \in D^+$ where $X \subseteq PathFuncs(C)$ and $P \in PathFuncs(C)$, there exists $K \in LKeys(C, P)$ such that $X \rightarrow K$ and $K \rightarrow P$ both hold so this schema is in OOFNF. $\triangle$

The next example contains a schema with redundancies. We would not expect this schema to be in OOFNF.

Example 3.5 Consider the following schema:
$course\{course\_number : string,\ \ student\_name : string,\ \ mums\_age : string\}$,
with the path functional dependencies $\{course\_number, student\_name\} \rightarrow Id$ and $student\_name \rightarrow mums\_age$.

The location keys $LKeys(course, P) = \{\{course\_name, student\_name\}\}$, for all $P \in PathFuncs(supplier)$. For the reduced functional dependency $student\_name \rightarrow mums\_age$ there is no K such that $K \in LKeys(course, mums\_age)$ where $student\_name \rightarrow K$ and $K \rightarrow mums\_age$ both hold so this schema is not in OOFNF. $\triangle$

In the above examples, the schemas consisted only of independent classes. We now consider an example with a dependent class.

Example 3.6 Consider the following example where each department has only one manager and the managers cannot be uniquely identified by their name or their salary. We would expect the OO schema to be in OOFNF.

$department\{dname : string, manager : manager\_class\}$

$manager\_class\{mname : string, salary : integer\}$

with path functional dependencies $D = \{\{dname\} \rightarrow manager,\ \{dname\} \rightarrow manager.mname, \{dname\} \rightarrow manager.salary\}$.

Intuitively, each path function in the dependent class is uniquely identified by the key of its parent independent class because all the properties are single valued. The new definition of LKeys confirms this intuition:

$LKeys(department,\ P) = \{\{dname\}\}$, for all $P \in PathFuncs(department)$.

Now for each reduced functional dependency $X \rightarrow P \in D^+$ where $X \subseteq PathFuncs(C)$ and $P \in PathFuncs(C)$, there exists $K \in LKeys(C, P)$ such that $X \rightarrow K$ and $K \rightarrow P$ both hold so the class supplier is in OOFNF. $\triangle$

Consider now a schema with inheritance. Inheritance is included in the PFD definition and doesn't need to be treated as a special case anywhere else.

Example 3.7 Consider the following example where each person has a unique name and a student has only one supervisor.

$person\{name : string\}$,

$teacher\{salary : integer\}$ ISA $person$,

$student\{supervisor : teacher\}$ ISA $person$,

with path functional dependencies $D = \{\{name\} \rightarrow salary,\ \ \{name\} \rightarrow supervisor, \{name\} \rightarrow supervisor.name,\ \ \{name\} \rightarrow supervisor.salary\}$.

The LKeys are

$LKeys(person, P) = \{\{name\}\}$, for all $P \in PathFuncs(person)$,

$LKeys(teacher, P) = \{\{name\}\}$, for all $P \in PathFuncs(teacher)$,

$LKeys(student, P) = \{\{name\}\}$, for all $P \in PathFuncs(student)$.

Now for each reduced functional dependency $X \rightarrow P \in D^+$ where $X \subseteq PathFuncs(C)$ and $P \in PathFuncs(C)$, there exists $K \in LKeys(C, P)$ such that $X \rightarrow K$ and $K \rightarrow P$ both hold so this schema is in OOFNF. $\triangle$

In summary, we have redefined location key and reduced functional dependency. We have demonstrated with examples that the new definitions give the expected results. We now prove that if a class is in OOFNF, the equivalent relation is in BCNF.

Consider a translation of a class into a relation. The class C becomes a relation R. Each property in C is an attribute in R. The SKeys(C) are the keys of R. The LKeys(C,P), for each path P, are also the keys of R. Each reduced PFD, that doesn't refer to Id, is a trivial FD. We say that R is the corresponding relation to C.

Example 3.8 Consider the schema described in example 3.1 with class

$supplier\{name : string, item : string, price : string\}$.

The $SKeys(supplier) = \{\{name, item\}\}$, and $LKeys(supplier, P) = \{\{name, item\}\}$, for all $P \in PathFuncs(supplier)$, and reduced PFDs $D = \{\{name, item\} \rightarrow Id,\ \ \{name, item\} \rightarrow price\}$.

The class supplier is translated to the relation supplier:

$supplier\{name : string, item : string, price : string\}$

with key $\{name, item\}$. The trivial FDs over $supplier$ are $\{\{name, item\} \rightarrow price\}$.

$\triangle$

Theorem 3.1 *Consider a class C that has only built-in typed properties. If this class C is in OOFNF then the corresponding relation R will be in BCNF.*

Proof Suppose that the theorem doesn't hold, i.e. C is in OOFNF and R is not in BCNF. Then there is a trivial FD $X \to Y$ such that X is not a key of R. It in turn means that there would be a reduced PFD $X \to Y$ such that $X \notin SKeys(C)$. Thus there would be no $K \in LKeys(C, Y)$ such that $X \to K$ and $K \to Y$ so the class wouldn't be in OOFNF. This is a contradiction. △

Conclusion

While the work in [5] is flawed, the intuition behind their work is good. They do need to separate independent and dependent classes (as they have). The definition of object oriented functional normal form (OOFNF) is based on Weddell's definition of path functional dependencies [9]. Path functional dependencies are defined on a data model that has single valued properties.

In this paper, we have fixed some of the flaws in their work. In a restricted data model (that has single-valued properties) we have redefined LKey and reduced functional dependency. We have proved the equivalence of our new definition with BCNF. The ideas in [5] may be applicable to a data model with multi-valued properties but the axioms from Weddell's work must be shown to hold in the less restrictive setting first.

In what way can this work be extended? The obvious next step is to extend this work to the less restrictive data model, i.e. a data model that allows multi-valued properties. This may involve extending Weddell's definitions, basing the work on a definition of functional dependency that holds in the less restrictive data model [3], or extending the work of others who have defined normalisation for the nested relational model [6, 7]. The papers [1, 8] also address normalisation in the OODB model.

References

[1] Eric Andonoff. Normalization of Object-Oriented Conceptual Schemes. In *Proceedings of the Conference on Advanced Information Systems Engineering (CAiSE)*, 1993.

[2] Gillian Dobbie. Towards normalization in object oriented databases. Technical Report CS-TR-96/16, Department of Computer Science, Victoria University of Wellington, 1996.

[3] Carmem S. Hara and Susan B. Davidson. Reasoning about Nested Functional Dependencies. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 1999.

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

[5] Hui Li, Maria E. Orlowska, and Chengfei Liu. A normal form on functional dependencies in object-oriented data models. In *Australasian Database Conference*, 1999.

[6] W.Y. Mok, Y. Ng, and D.W. Embley. A normal form for precisely characterising redundancy in nested relations. *ACM Transactions on Database Systems*, 21(1):77–106, 1996.

[7] Z.M Ozsoyoglu and L. Yuan. A new normal form for nested relations. *ACM Transactions on Database Systems*, 12(1):111–136, 1987.

[8] Zahir Tari, John Stokes, and Stefano Spaccapietra. Object normal forms and dependency constraints for object-oriented schemata. *ACM Transactions on Database Systems*, 22(4):513–569, 1997.

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