Ulmer Informatik-Berichte, Nr. 94-06, June 1996

Incorporating record subtyping into a relational data model

Christian Kalus, Peter Dadam
Faculty of Computer Science, Dept. Databases and Information Systems
University of Ulm, D 89069 Ulm, Germany
e-mail: {kalus,dadam}@informatik.uni-ulm.de

Abstract

Most of the current proposals for new data models support the construction of heterogeneous sets. One of the major challenges for such data models is to provide strong typing in the presence of heterogenity. Therefore the inclusion of as much as possible information concerning legal structural variants is needed. We argue that the shape of some part of a heterogeneous scheme is often determined by the contents of some other part of the scheme. This relationship can be formalized by a certain type of integrity constraint we have called *attribute dependency*. Attribute dependencies combine the expressive power of general sums with a notation that fits into relational models. We show that attribute dependencies can be used, besides their application in type and integrity checking, to incorporate record subtyping into a relational model. Moreover, the notion of attribute dependency yields a stronger assertion than the traditional record subtyping rule as it considers some refinements to be caused by others.

To examine the differences between attribute dependencies and traditional record subtyping and to be able to predict how attribute dependencies behave under transformations like query language operations we develop an axiom system for their derivation and prove it to be sound and complete. We further investigate the interaction between functional and attribute dependencies and examine an extended axiom system capturing both forms of dependencies.

1 Introduction

The relational model as defined by Codd [Codd70] forms the base of most contemporary data models. It constructs a database as a set of relations, a relation being defined over a set of attributes called its scheme. The instance of a relation is a set of tuples where each tuple is a mapping from the scheme attributes to values of given (atomic) domains.
The constraint of homogeneity, i.e. the fact that all tuples of a relation are defined over the same set of attributes, does often not meet the intuition of a relation as a container of *related* entities. Take an address (e.g. of a person's record) as a simple example. Each address comprises a zip code and a town. The town-local part of the address may be either a post-office box number or a street and, if it is a street, it is sometimes followed by a house number, sometimes not. This little example already exhibits many facets: ZipCode and Town are *unconditioned* or *homogeneous* components as they are always present. The town-local part is a *disjoint union* of PostOfficeBoxNumber and Street while Street may be accompanied by the *optional attribute* HouseNumber.
Another form of attribute relationship can be motivated by the "electronic part" of an address which is composed of a telephone number, a FAX number, and an electronic mail address. At least one of these three attributes should be present to constitute an electronic address, but two or all three of the attributes are allowed too. This *non-disjoint union* is another relationship type between attributes, and many other forms of relationships can be found in addition.

The relationships mentioned above are based on the pure existence of attributes. I.e. from the sole existence/absence of a certain attribute we draw conclusions about the existence or absence of other attributes. In addition to these *existence-based attribute relationships* there occur *value-based attribute relationships* which take the influence of values (in some attributes) on the existence of (other) attributes into account.

Take an employee entity possessing the attributes salary and job-type as an example. In addition,

- if the value of job-type is 'secretary' then the attributes typing-speed and foreign-languages are present.
- if the job-type is 'software engineer' the employee is further described by the products he is in charge of and the programming languages he knows.
- if the job-type is 'salesman', he possesses a sales commission and also the products he is in charge of[1].

Another example is that the existence of a maiden name is determined by the appropriate values in the attributes sex and marital status.

The overall aim of the model of flexible relations is to bridge the gap between semantic data models and operational data models. Therefore it captures the attribute relationships described above, yet it utilizes a minimum of (generic) constructs to preserve the elegance of the relational model.
The paper is focussing on value-based attribute relationships. We will introduce a notation we have called *attribute dependencies* that enables us to model these relationships as integrity constraints. We will motivate their usage, particularly their ability to model a strong notion of subtyping. Besides their employment in record subtyping and their connection to semantic type constructs, the benefit of attribute dependencies in type and integrity checking and in host language coupling is discussed. The formal behaviour of attribute dependencies will be described by a sound and complete axiom system for their derivation.

The rest of the paper is structured as follows: Section 2 introduces some basic notations of the model of flexibe relations needed as an environment in which attribute dependencies are to be integrated. The different purposes which attribute dependencies serve in our model, including subtyping, are discussed in section 3. In section 4 an axiom system for the derivation of attribute dependencies is developed and shown to be sound and complete. In addition, an extended axiom system capturing both functional and attribute dependencies is developed. Section 5 compares our approach to related ones, while section 6 concludes with a summary and an outlook.

[1] Note that we mix attributes with atomic values (like salary) and attributes with composed values (like products) as this distinction is not subject of our discussion.

2 Attribute Dependencies

2.1 Basic notations of the model of flexible relations

The elegance of the relational model is mainly due to the fact that it gets along with a single type constructor. To preserve this elegance as much as possible, we looked for an extended type constructor enabling us to describe the various variant (and also non-variant) structures in a single, generic fashion. It is obvious that specifying a scheme as a set of attributes, as the relational model does, is not expressive enough to model arbitrary attribute relationships. To achieve this goal we enhanced the scheme notation in the following way: a scheme is now composed of a set of attributes accompanied by a cardinality constraint in the form of two integer values determining how many components of the set have *at least* to be taken and how many components of the set are allowed *at most*. If we describe this construct as a three-tuple

< at-least value, at-most value, set of attributes >

then the various constructs introduced in section 1 can be expressed in the following way[2]

- a traditional relational scheme with attributes A_1 , ... , A_n is denoted by $< n, n, \{ A_1 , ... , A_n \} >$, i.e. at least n and at most n (and therefore exactly n) of the attributes have to be present.
- a disjoint union of attributes A_1 , ... , A_n is modeled by $< 1, 1, \{ A_1 , ... , A_n \} >$ telling that exactly one of the attributes may appear.
- a non-disjoint union of attributes A_1 , ... , A_n is described by $< 1, n, \{ A_1 , ... , A_n \} >$, i.e. the electronic address of section 1 is expressed by < 1, 3, { telephone-number, FAX-number, email-address } >.

The above notation is not completely satisfying yet: a union might appear as only a part of a scheme, the variants in a union do not need to be single attributes but can be relational schemes, variants again, and so on. Therefore we have to extend our notation allowing the set components to be either single attributes or again three-tuples of our notation. This final version of a *flexible scheme* is presented by a more abstract example.

---

[2] Let as usual be $\mathcal{U}$ the universe of attributes, A, B ... and A_i be single attributes, and V, ... , Z be attribute sets. Let XY denote the union of the attribute sets X and Y and treat attributes as singleton attribute sets when sets of attributes are expected.

Example 1 An application demanding tuples with attributes A and B (unconditioned), either attribute C or D (i.e. a disjoint variant between C and D) and "some" of E, F and G (a non-disjoint union of E, F and G) yields the following flexible scheme FS

$$FS = < 4, 4, \{ A, B, < 1, 1, \{ C, D \} > , < 1, 3, \{ E, F, G \} > \} >$$

□

A flexible scheme is a very compact notation. For the purpose of a basic understanding one can unfold a flexible scheme yielding the allowed attribute combinations. As this unfolding can be interpreted as building the *disjunctive normal form* of a flexible scheme FS, we will refer to it as *dnf*(FS). Forming the DNF of the scheme of example 1 yields[3]

dnf(FS) = { ABCE, ABDE, ABCF, ABDF, ABCG, ABDG, ABCEF, ABDEF, ABCEG, ABDEG, ABCFG, ABDFG, ABCEFG, ABDEFG }

Now it is easy to define the domain of a flexible scheme. If *Tup*(X) denotes the set of tuples for a given attribute set X , then $dom(FS) = \bigcup_{X \in dnf(FS)} Tup(X)$. A *flexible relation* FR can then be defined as a two-tuple FR = < FS, inst > with *scheme*(FR) = FS being a flexible scheme and *inst*(FR) = inst being the instance of the relation, a finite set of tuples satisfying $inst(FR) \subset dom(scheme(FR))$. As a flexible scheme does not uniquely determine the shape of its tuples we assume the existence of a function *attr*(t) yielding the attribute set X , tuple t is defined on (of course $attr(t) \in dnf(FS)$, if $t \in dom(FS)$).

2.2 Definition of attribute dependencies

Up to now a flexible scheme considers existential relationships of attributes and determines thus the basic shape of tuples and instances. Value-based constraints are not yet taken into account, a flexible scheme is therefore, following the notation of [PDGV89], a *primitive* scheme and a flexible relation only a *possible* relation (instance). The examples in section 1 have shown that flexible relations demand, besides the known types of constraints, a certain class of constraints concerning the

[3] Note that this unfolded version of a flexible scheme corresponds to the "set of objects" idea of Ed Sciore [Scio80] (see also [Maie83,chapter 12]). The little example above should suffice as a motivation to find a compact description for variant schemes.

variant structure of tuples. Referring to the *job-type*-example of section 1 we may say that the value of the attribute job-type determines the existence of the attributes in Y = { typing-speed, foreign-languages, products, programming-languages, sales-commission } in the way that

t(job-type) = 'secretary' $\rightarrow$ attr(t) $\cap$ Y = { typing-speed, foreign-languages }
t(job-type) = 'software engineer' $\rightarrow$ attr(t) $\cap$ Y = { products, programming-languages }
t(job-type) = 'salesman' $\rightarrow$ attr(t) $\cap$ Y = { products, sales-commission }

To prepare the formal definition of an attribute dependency consider the following points. While in the example above there is only one determining attribute (job-type), in general there may be several ones (take sex and marital-status determining the existence of a maiden name). Therefore we should say that the contents in the attribute set X determines which attributes in the attribute set Y exist. This general assertion can be refined by considering the legal variants explicitely. Each variant consists of an attribute set $Y_i \subseteq Y$ (i=1..n, n being the number of variants) and is determined by a set of values $V_i \subseteq Tup(X)$ with the obvious meaning that the attribute set Y_i occurs in a tuple t whenever $t[X] \in V_i$. When there is no V_i such that $t[X] \in V_i$ then it is intuitive to demand that tuple t does not possess any attribute of Y. Considering this we obtain the definition of an *attribute dependency* [4]

Definition 2.1 "explicit attribute dependency"

An explicit attribute dependency EAD has the syntactical form

$$EAD = < X \xrightarrow{exp.attr} Y , \{ V_1 \xrightarrow{exp.attr} Y_1 , \ldots , V_n \xrightarrow{exp.attr} Y_n \} >$$

where $X \subset \mathcal{U}$, $V_i \subseteq Tup(X)$ (i = 1 .. n),
$Y \subset \mathcal{U}$, $Y_i \subseteq Y$ (i = 1 .. n),
$i \neq j \rightarrow V_i \cap V_j = \varnothing$ (i,j = 1 .. n)

A flexible relation FR is said to satisfy the explicit attribute dependency EAD if

$$\forall t \in inst(FR) : X \subseteq attr(t) \rightarrow ((\exists i : t[X] \in V_i) \rightarrow attr(t) \cap Y = Y_i)$$
$$\wedge \; X \subseteq attr(t) \rightarrow ((\forall i : t[X] \notin V_i) \rightarrow attr(t) \cap Y = \varnothing)$$

□

[4] In section 4 we will use a slightly modified definition. To distinguish both we call the following definition an *explicit attribute dependency* .

Example 2 The *job-type*-example is formulated in the EAD-notation by

$$
\begin{aligned}
&< \{\text{job-type}\} \xrightarrow{\text{exp.attr}} \{\text{typing-speed,foreign-languages,products,programming-languages,sales-commission}\} ,\\
&\quad \{ < \text{job-type : 'secretary'} > \xrightarrow{\text{exp.attr}} \{ \text{typing-speed, foreign-languages} \} ,\\
&\quad\ < \text{job-type : 'software engineer'} > \xrightarrow{\text{exp.attr}} \{ \text{products, programming-languages} \} ,\\
&\quad\ < \text{job-type : 'salesman'} > \xrightarrow{\text{exp.attr}} \{ \text{products, sales-commission} \} \ \ \}\\
&>
\end{aligned}
$$

□

3 Usage of attribute dependencies

There are two main streams motivating the use of attribute dependencies. The first one is to integrate semantic type constructs into an operational data model, thus bridging the gap between semantic and operational data models. The second reason is to show that attribute dependencies may be used to incorporate subtyping into a relational data model. Here we will go even further and we are going to stress the point that the traditional subtyping rule does not manage causal relationships between type parts correctly.

3.1 Mapping of entity-relationship concepts onto flexible relations

Specialization is one of the enhanced entity relationship concepts [ElNa89, chapter 15]. A specialization that is encoded in the entity itself is called a *predicate defined specialization*. If one replaces the predicate p_i of the i-th specialization by its extension V_i [5], i.e. $V_i = \{ v \mid p_i(v) \text{ is true} \}$, then an attribute dependency is a one-to-one mapping of a predicate defined specialization. ER models further divide specialization into *disjoint* versus *overlapping* subclasses and *total* versus *partial* subclasses. This classification can be inferred from attribute dependencies as well: the variants of an attribute dependency are *disjoint* if $Y_i \cap Y_j = \varnothing$ $(i \neq j)$ and they are *total* if $\bigcup_{i=1..n} V_i = \mathit{Tup}(X)$. The benefit of mapping these ER constructs onto the model of flexible relations is that they can now be exploited operationally.

[5] The meaning of V_i is explained in definition 2.1 .

The most important operational use of attribute dependencies is their application in type-checking, which is a central point of our model. Flexible schemes do serve this purpose already better than relational schemes as existential attribute relationships are already captured by them[6]. However, value-based dependencies cannot be type-checked by flexible schemes. For example, there is no way to construct a flexible scheme which would reject the tuple

< .. jobtype : 'salesman', typing-speed : high, foreign-languages : { french, russian } >

as { ... , jobtype, typing-speed, foreign-languages } is a valid attribute combination. The fact that jobtype = 'salesman' requires different attributes to be present has to be formulated with the attribute dependency of example 2.
Type checking based on attribute dependencies is initiated during insertion, update[7], and data retrieval. In the context of retrieval two tasks have to be acomplished: A precise type has to be given to the query result and redundant type guard operations have to be eliminated [8].

3.2 Semantic preserving subtyping through attribute dependencies

At first glance a predicate defined specialization can as well be described by the traditional record subtyping rule (see [CaWe85] among many others)

$$\frac{t_i \leq u_i \ (i=1..n)}{[\ a_1 : t_1\ ,\ \dots\ ,\ a_n : t_n\ ,\ \dots\ ,\ a_m : t_m\ ] \leq [\ a_1 : u_1\ ,\ \dots\ ,\ a_n : u_n\ ]}$$

[6] The type-checking of explicit attribute dependencies can be demonstrated with the flexible scheme of example 1: Although the occuring attributes are { A, B, C, D, E, F, G } , the tuple < A:a, B:b, C:c, D:d, E:e, F:f, G:g > would be rejected as attributes C and D exclude each other.

[7] While there are no further type-related consequences when the salary of an employee is updated, the change of his jobtype causes a type change, too.

[8] When dealing with heterogeneous sets, type guards check for the presence of attributes. The presence or absence of the concerned attributes may be inferred if the determining attributes of an attribute dependency are involved in a selection formula (jobtype == 'salesman'). In this case the type guard would be redundant. In addition, the type of the query result may be restricted to the type of those variants satisfying the selection formula.

Let us first show that this inclusion rule can be expressed with an attribute dependency. Therefore, consider a flexible scheme FS with *attr*(FS) = W and let

$$EAD = < X \xrightarrow{exp.attr} Y , \{ V_1 \xrightarrow{exp.attr} Y_1 , ... , V_n \xrightarrow{exp.attr} Y_n \} >$$

be an attribute dependency for FS. Then the corresponding supertype contains the attributes W - Y and the domain of X consists of *Tup*(X), i.e. the domain of X is unrestricted in the supertype. Further we can derive from EAD that there are n subtypes possessing the attributes $(W - Y) \cup Y_i$, having the domain of X restricted to V_i (i=1..n). We may therefore say that attribute dependencies incorporate record subtyping into a relation-based model [9].

Now, what is the benefit of using attribute dependencies instead of the traditional subtyping rule? This rule is obviously sufficient to state that *secretary*, *software engineer* and *salesman* type are subtypes of a more general *employee* type (see figure 3.1 below)

employee_type = < salary : float, jobtype : { 'secretary', 'software engineer', 'salesman' } >

secretary_type = < salary : float, jobtype : { 'secretary' }, typing-speed : ... , foreign-languages : ... >

softw_eng_type = < salary : float, jobtype : {'software engineer' }, products : .. , programming-languages : .. >

salesman_type = < salary : float, jobtype : {'salesman' }, products : ... , sales-commission : ... >

Fig. 3.1 Employee type and its predicate defined subtypes

Note that for each of the three subtypes there are two type changes causing the subtype relation: The domain of jobtype is restricted and some attributes are added to the subtypes. These simultaneous type changes are considered to be purely accidental by the record subtyping rule. The type

< salary : float >

is therefore treated as a valid supertype of the subtypes presented above, although the connection between the determining attribute jobtype and the subtypes is destroyed. To prevent this from happening or at least to notify the loss of this connection, it is necessary to treat these type changes as causal related, like attribute dependencies do.

[9] Although we do not talk about methods in this paper, it is easy to see that a behaviourally object oriented system could be put upon our model and utilize the described subtype relationship.

3.3 Further usage of attribute dependencies

A last usage, which shall only be sketched here, is that attribute dependencies are an encoding of general sums (see [MacQ86] as an entry point) to make this type construct fit into a relational model. This equivalence can be exploited when embedding of flexible relations into programming language is discussed. It can be shown that a flexible scheme can be translated into an appropriate programming language type (e.g. a variant record in PASCAL) if each existential attribute relationship is accompanied by an attribute dependency. If necessary, this can be obtained by introducing artificial attribute dependencies with artificial determining attributes.

The applications discussed in this section pose different requirements on attribute dependencies. In some cases, like insertion, whole tuples of a flexible relation are considered. Here, it is sufficient to apply the attribute dependencies specified in the scheme. But most applications, like update, retrieval or programming language embedding, are referring only to parts of a tuple or to tuples which may even have been transformed by (query language) operations. Therefore it is also necessary to know how attribute dependencies behave under transformations. This question is also the central point when attribute dependencies are exploited for (semantic preserving) subtyping. Facing the transformation problem differently, we have to examine which valid attribute dependencies may be derived from given ones. To answer this question we will develop an axiom system for the implication of attribute dependencies. This is done in the next section.

4 An axiom system for attribute dependencies

Before we define the axiom system we slightly modify the definition of an attribute dependency. This is only done for the sake of readability and to better illustrate the similarity to other forms of dependencies, but does not change the intention.
From definition 2.1 we can derive that, given an explicit attribute dependency $< X \xrightarrow{exp.attr} Y \ldots >$, whenever two tuples t_1, t_2 agree on X, then they possess the same subset of Y as attributes. Therefore we can find another definition for attribute dependencies which serves better for the purpose of finding an appropriate axiom system.

Definition 4.1 "attribute dependency"

Let X, Y $\subset$ $\mathcal{U}$. A flexible relation FR is said to satisfy the attribute dependency $X \xrightarrow{attr} Y$ if $\forall t_1, t_2 \in inst(FR)$

$X \subseteq attr(t_1) \wedge X \subseteq attr(t_2) \wedge t_1[X] = t_2[X] \rightarrow attr(t_1) \cap Y = attr(t_2) \cap Y$

□

The axiom system $\mathcal{A}$ that manages attribute dependencies consists of the following four rules:

(A1)	$X \xrightarrow{attr} YZ$:- $X \xrightarrow{attr} Y$	(projectivity)
(A2)	$\{ X \xrightarrow{attr} Y , X \xrightarrow{attr} Z \}$:- $X \xrightarrow{attr} YZ$	(additivity)
(A3)	$\varnothing$:- $X \xrightarrow{attr} Y$ if $Y \subseteq X$	(reflexivity)
(A4)	$X \xrightarrow{attr} Y$:- $XZ \xrightarrow{attr} Y$	(left augmentation)

A remarkable point about this rule system is that transitivity is not valid for attribute dependencies. This stems from the fact that we do not draw any conclusion about the contents of the determined attributes. Nevertheless other rules, like the augmentation ($X \xrightarrow{attr} Y$:- $XZ \xrightarrow{attr} YZ$) or the right augmentation ($X \xrightarrow{attr} Y$:- $X \xrightarrow{attr} XY$) are valid, too. The following theorem tells us that they can be derived from $\mathcal{A}$ and that their inclusion would therefore not enhance the power of the axiom system[10].

Theorem 4.1 $\mathcal{A}$ is a sound, complete and non-redundant system of axioms for the implication of attribute dependencies.

□

[10] The proofs of all theorems are presented in the appendix.

Note that all rules could have been defined for explicit attribute dependencies as well. For example, the additivity rule would be

$$\{ < X \xrightarrow{\text{exp.attr}} Y , \{ V_{11} \xrightarrow{\text{exp.attr}} Y_{11} , \ldots , V_{1n} \xrightarrow{\text{exp.attr}} Y_{1n} \} > ,$$
$$< X \xrightarrow{\text{exp.attr}} Z , \{ V_{21} \xrightarrow{\text{exp.attr}} Z_{21} , \ldots , V_{2m} \xrightarrow{\text{exp.attr}} Z_{2m} \} > \} \text{ :-}$$
$$< X \xrightarrow{\text{exp.attr}} YZ , \{ V_{11} \cap V_{21} \xrightarrow{\text{exp.attr}} Y_{11}Z_{21} , \ldots , V_{1n} \cap V_{2m} \xrightarrow{\text{exp.attr}} Y_{1n}Z_{2m} \} >$$

This lengthy definition hampers of course the readability, thus making the abbreviated definition of attribute dependencies more favorable for our purpose. Nevertheless we stress again that the presented axiom system works for explicit attribute dependencies as well.

Example 3 Suppose a query containing a selection with the formula "salary > 5000 AND jobtype = 'secretary' " followed by a type guard checking for the presence of the attribute typing-speed. The redundancy of the type guard can be shown by the following derivation based on the explicit attribute dependency defined in example 2 (recapitulated below) :

$$< \{\text{job-type}\} \xrightarrow{\text{exp.attr}} \{\text{typing-speed,foreign-languages,products,programming-languages,sales-commission}\} ,$$
$$\{ < \text{job-type : 'secretary'} > \xrightarrow{\text{exp.attr}} \{ \text{typing-speed, foreign-languages} \} ,$$
$$< \text{job-type : 'software engineer'} > \xrightarrow{\text{exp.attr}} \{ \text{products, programming-languages} \} ,$$
$$< \text{job-type : 'salesman'} > \xrightarrow{\text{exp.attr}} \{ \text{products, sales-commission} \} \ \}$$
$$>$$

Projecting the right side of the given EAD onto { typing-speed } yields (cf. rule (A1))

$$< \text{job-type} \xrightarrow{\text{exp.attr}} \text{typing-speed} , \{ < \text{job-type : 'secretary'} > \xrightarrow{\text{exp.attr}} \text{typing-speed} \} >$$

Augmenting the left side of this EAD with the attribute salary yields (cf. rule (A4))

$$< \{ \text{job-type, salary} \} \xrightarrow{\text{exp.attr}} \text{typing-speed} , \{ < \text{job-type : 'secretary' , salary : s} > \xrightarrow{\text{exp.attr}} \text{typing-speed} \} >$$

where s is an arbitrary value of *dom*(salary). We may conclude that the presence of the attribute typing-speed can be deduced from the selection formula and that the type guard is therefore redundant.

□

4.2 An extended axiom system capturing functional and attribute dependencies

There are several reasons why one should regard functional dependencies together with attribute dependencies. The first and most practical reason origins from the discussion how to embed flexible relations into programming languages. Take PASCAL as an example: Although its variant record type resembles attribute dependencies, there are some syntactic restrictions that have to be obeyed. One of them is that only a single attribute may appear as determinant of a variant record. Suppose now an attribute dependency $X \xrightarrow{attr} Y$ with X consisting of at least two attributes. There is an intuitive way to circumvent the aforementioned syntactic restriction: Introduce an artificial attribute A, replace $X \xrightarrow{attr} Y$ by $A \xrightarrow{attr} Y$ and make the value of A dependent on the value of X, i.e. extend the constraints by $X \xrightarrow{func} A$. The validity of this and other replacements may be verified by the aid of a rule system combining functional and attribute dependencies.
From the more theoretical point of view it is interesting to consider the interaction between both forms of dependencies. Especially, there are two of the rules in $\mathfrak{A}$ that are not derivable due to the (intended) weak consequence of an attribute dependency. Namely, X does not only determine the existence of Y if Y is a subset of X (see reflexivity rule (A3) above), but also its value (see reflexivity rule (F1) below). A combined axiom system will therefore not only show the connections between both forms of dependencies but will also uncover those rules of $\mathfrak{A}$ that are really non-redundant.

We, therefore, adapt the notion of functional dependencies to fit into the model of flexible relations. The adaption simply consists of the adding of a type guard "$X \subseteq attr(t)$", as the access of values must be preceded by such a type guard in our model. The axiom system, consisting of the reflexivity rule, the transitivity rule, and the augmentation rule (see (F1), (F2) and (F3) below), is borrowed from [Ullm88,p.384ff]. Its soundness, completeness and non-redundancy is not affected by the adaption of the definition to our model.

Definition 4.2 "functional dependency (adapted to flexible relations)"

Let $X,Y \subset \mathcal{U}$. A flexible relation FR is said to satisfy the functional dependency $X \xrightarrow{\text{func}} Y$ if $\forall t_1, t_2 \in \mathit{inst}(FR)$

$$X \subseteq \mathit{attr}(t_1) \wedge X \subseteq \mathit{attr}(t_2) \wedge t_1[X] = t_2[X] \rightarrow$$
$$Y \subseteq \mathit{attr}(t_1) \wedge Y \subseteq \mathit{attr}(t_2) \wedge t_1[Y] = t_2[Y]$$

□

The combined axiom system $\mathcal{AF}$ for functional and attribute dependencies consists of the following seven rules:

(AF1)	$X \xrightarrow{\text{func}} Y$:- $X \xrightarrow{\text{attr}} Y$	(subsumption)
(AF2)	{ $X \xrightarrow{\text{func}} Y$, $Y \xrightarrow{\text{attr}} Z$ } :- $X \xrightarrow{\text{attr}} Z$	(combined transitivity)
(A1)	$X \xrightarrow{\text{attr}} YZ$:- $X \xrightarrow{\text{attr}} Y$	(projectivity)
(A2)	{ $X \xrightarrow{\text{attr}} Y$, $X \xrightarrow{\text{attr}} Z$ } :- $X \xrightarrow{\text{attr}} YZ$	(additivity)
(F1)	$\emptyset$:- $X \xrightarrow{\text{func}} Y$ if $Y \subseteq X$	(reflexivity)
(F2)	$X \xrightarrow{\text{func}} Y$:- $XZ \xrightarrow{\text{func}} YZ$	(augmentation)
(F3)	{ $X \xrightarrow{\text{func}} Y$, $Y \xrightarrow{\text{func}} Z$ } :- $X \xrightarrow{\text{func}} Z$	(transitivity)

Theorem 4.2 $\mathcal{AF}$ is a sound, complete and non-redundant system of rules for the implication of functional and attribute dependencies.

□

Referring to the motivation for this combined rule system, one can see that the pragmatic "work-around" for PASCAL's variant record type is valid (see the combined transitivity rule (AF2)). Secondly, we could save two rules still needed in $\mathcal{A}$ to produce a complete axiom system. The removed rules are

(A3)	$\emptyset$:- $X \xrightarrow{\text{attr}} Y$ if $Y \subseteq X$	(reflexivity)
(A4)	$X \xrightarrow{\text{attr}} Y$:- $XZ \xrightarrow{\text{attr}} Y$	(left augmentation)

The derivation sequences are the following: Applying the subsumption rule (AF1) to the reflexivity rule (F1) yields rule (A3). To derive (A4) we use $XZ \xrightarrow{\text{func}} X$ (reflexivity rule (F1)) and $X \xrightarrow{\text{attr}} Y$ (given), and apply the combined transitivity rule (AF2) to both, yielding rule (A4).

4.3 Attribute dependencies versus record subtyping - the impact of transformations

At the end of this section we want to discuss the differences and similarities of attribute dependencies and traditional record subtyping. To do so, we sketch, with the aid of the developed axiom system, how transformations affect attribute dependencies. As the formal description of the algebra for the model of flexible relations is beyond the scope of this paper, we will rely upon well-known algebraic operator, providing the intuitive meaning in our model, too.

The most remarkable difference between record subtyping and attribute dependencies shows the project operator. While record subtyping tells us that any projection yields a valid supertype [ScSc90], two cases have to be discriminated for attribute dependencies: Suppose that a flexible relation is to be projected onto the attribute set X. As there is no rule telling us that an attribute dependency may hold if attributes at its left side are omitted, all $V \xrightarrow{attr} W$ with $V \not\subseteq X$ are invalidated. If on the other hand $V \subseteq X$ then the projection rule tells us that $V \xrightarrow{attr} W \cap X$ holds in the projection.

The two notions of subtyping perform similar when the result "enlarges" the input relation(s). This holds e.g. for the extension operator and the cartesian product. The behaviour of attribute dependencies under algebraic transformations can be summarized as follows:

Theorem 4.3 Let *ads*(FR) be the set of attribute dependencies that hold in the flexible relation FR. The following rules describe the propagation of attribute dependencies:

(1) $ads(FR_1 \times FR_2) = ads(FR_1) \cup ads(FR_2)$

(2) $ads(\pi_X(FR)) = \{ V \xrightarrow{attr} W \cap X \mid V \xrightarrow{attr} W \in ads(FR) \wedge V \subseteq X \}$

(3) $ads(\sigma_F(FR)) = ads(FR)$

(4) $ads(FR_1 \cup FR_2) = \varnothing$

(5) $ads(FR_1 - FR_2) = ads(FR_1)$

□

The theorem shows that besides the projection the union operator causes a problem, too. First of all note that without appropriate precautions no dependency at all holds in the result of a union, as one cannot decide from which input relation the tuples do come from. To make dependencies hold, one has to tag both input relations before performing the union. The tagging can be realized with the extension operator $\varepsilon_{A:a}(FR)$, which extends each tuple of FR by attribute A with value 'a'.

The left augmentation rule allows us to replace any $X \xrightarrow{attr} Y$ occuring in one of the input relations by $AX \xrightarrow{attr} Y$ in its extended counterpart. The extended attribute dependencies now remain valid in the result relation, i.e. we obtain

(6) $ads((\varepsilon_{A:a1} (FR_1)) \cup (\varepsilon_{A:a2} (FR_2))) =$

$$\{ AX \xrightarrow{attr} Y \mid X \xrightarrow{attr} Y \in ads(FR_1) \wedge X \xrightarrow{attr} Y \in ads(FR_2) \}$$

(7) $ads((\varepsilon_{A1:a1} (\varepsilon_{A2:unique()} (FR_1))) \cup (\varepsilon_{A1:unique()} (\varepsilon_{A2:a2} (FR_2)))) =$

$$\{ A_1X \xrightarrow{attr} Y \mid X \xrightarrow{attr} Y \in ads(FR_1) \} \cup \{ A_2X \xrightarrow{attr} Y \mid X \xrightarrow{attr} Y \in ads(FR_2) \}$$

5 Related work

The concept of subtyping is present in any object-oriented data model [AtBa89]. To obey the desirable closure property, these data models should discuss how the subtype relation is affected by algebraic tranformations. This has been done e.g. for the COCOON model in [ScSc90] and for the ENCORE model in [ShZd90]. Differences to our notion of record subtyping have been discussed in section 3.2, the comparison of the behaviour under transformations is contained in section 4.3.
From the viewpoint of data dependencies, several attempts have been made to consider value-oriented dependencies in the presence of null values ([Vass80], [Lien79]), but considering a merely existential consequence without any value-oriented assertion seems to be a novelty in the context of an operational data model.

In [ElNa89] predicate defined specializations are used for the decomposition of an entity subtree (an entity type with its subclasses) into several relations. As variant structures are naturally incorporated in our model, there is no need to decompose a flexible relation along an attribute dependency. However, in the context of the traditional relational model decomposition along an attribute dependency is reasonable. In this case our formal treatment of attribute dependencies enables us to verify the correctness and losslessness of the decomposition strategies described in [ElNa89, chapter 15.2.1].
An approach which pursuits the idea of [ElNa89] to decompose an entity subtree into a master relation and depending relations containing the variant information is the "multirelation" model of Ahad and Basu [AhBa91]. They improve the decomposition by keeping track of the connection between the master relation and the depending relations so that the restoration of the complete information can be automated. The recording of the connection between master and depending relation is done via so-called "image attributes", attributes possessing relation names as their domain. Image attributes can be regarded as a special case of an attribute dependency using a single artificial attribute as determinant, this approach is therefore completely covered by our results.

6 Summary and outlook

In this paper we have motivated attribute dependencies as constraints naturally arising when variant structures are considered. It turned out that they can be used to incorporate record subtyping in a relational model, yielding an even stronger notion of subtyping, as attribute dependencies consider causal connections between type refinements. In addition we could show that the several forms of specialization arising in (enhanced) entity relationship models can be one-to-one mapped onto attribute dependencies, with the benefit that they can be operationally employed in type and integrity checking.
Our approach to model these features as a dependency allowed us develop an axiom system for their derivation. The axiom system has been shown to be sound, non-redundant and complete, which enables us to precisely predict the effect of arbitrary transformations (like query language operations) on attribute dependencies. Furthermore the connection between attribute and functional dependencies has been discussed and an extended axiom system capturing both forms of dependencies has been evaluated.
Although attribute dependencies were regarded in the context of the model of flexible relations they are rather loosely tied to particularities of this model (see section 2). Thus there seem to be no major problems to integrate attribute dependencies into other data models supporting variant structures or appropriate null values.

In this paper the connection between attribute dependencies and subtyping has been shown. However, the second axiom system presented here, capturing both functional and attribute dependencies, has been motivated differently (see section 4.2). Additional work should be put on the question if this combined rule system can be exploited to put further semantics into the subtype relationship.

Acknowledgements

I would like to thank Marc Scholl and Franz Schweiggert for carefully reading this paper. Also, thanks go to Klaus Gassner for lots of fruitful discussions.

Bibliography

AhBa91 Rafiul Ahad, Amit Basu, "ESQL: A Query Language for the Relation Model Supporting Image Domains", 7th Int. Conf. on Data Engineering, Kobe, Japan, April 1991, pp. 550 - 559

AtBa89 M. Atkinson, F. Bancilhon et al., "The Object-Oriented Database System Manifesto", 1st Int. Conf. on Deductive and Object-Oriented Databases, Kyoto, Japan, December 1989, pp. 40 - 57

CaWe85 Luca Cardelli, Peter Wegner, "On Understanding Types, Data Abstraction, and Polymorphism", ACM Computing Surveys, vol. 17, no. 4, December 1985, pp. 471 - 522

Codd70 E.F. Codd, "A Relational Model for Large Shared Data Bases", Communications of the ACM, vol. 13, no. 6, June 1970

ElNa89 Ramez Elmasri, Shamkant B. Navathe, "Fundamentals of Database Systems", Benjamin/Cummings Publishing Company, 1989

Lien79 Y. E. Lien, "Multivalued Dependencies with Null Values in Relational Databases", 5th Int. Conf. on Very Large Data Bases, Rio de Janeiro, Brazil, 1979, pp. 155 - 168

MacQ86 David MacQueen, "Using Dependent Types to Express Modular Structure", 13th Annual ACM Symposium on Principles of Programming Languages, 1986, pp. 277 - 286

Maie83 David Maier, "The Theory of Relational Databases", Computer Science Press, 1983

PDGV89 J. Paredaens, P. DeBra, M.Gyssens, D. VanGucht, "The Structure of the Relational Database Model", EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1989

Scio80 E. Sciore, "The Universal Instance and Database Design", Doctoral diss., Princeton Univ., Princeton, NJ, October 1980

ScSc90 Marc H. Scholl, Hans-Jörg Schek, "A Relational Object Model", Third Int. Conf. on Database Theory, LNCS 470, Paris, France, December 1990, pp. 89 - 105

ShZd90 Gail M. Shaw, Stanley B. Zdonik, "A Query Algebra for Object-Oriented Databases", 6th Int. Conf. on Data Engineering, Los Angeles, USA, February 1990, pp. 154 - 162

Ullm88 Jeffrey D. Ullman, "Principles of Database and Knowledge-Base Systems, Volume 1", Computer Science Press, 1988

Vass80 Yannis Vassilou, "Functional Dependencies and Incomplete Information", 6th Int. Conf. on Very Large Data Bases, Montreal, Canada, 1980, pp. 260 - 269

Appendix

Theorem 4.1 Let $\mathfrak{A}$ be the following system of axioms:

(A1)	$X \xrightarrow{attr} YZ$:- $X \xrightarrow{attr} Y$	(projectivity)
(A2)	$\{ X \xrightarrow{attr} Y , X \xrightarrow{attr} Z \}$:- $X \xrightarrow{attr} YZ$	(additivity)
(A3)	$\emptyset$:- $X \xrightarrow{attr} Y$ if $Y \subseteq X$	(reflexivity)
(A4)	$X \xrightarrow{attr} Y$:- $XZ \xrightarrow{attr} Y$	(left augmentation)

$\mathfrak{A}$ is a sound, complete and non-redundant system of axioms for the implication of attribute dependencies.

Proof.

Soundness: Let FR be a flexible relation. Let t_1, t_2 be two arbitrary tuples of *inst*(FR) satisfying the premise of the attribute dependency implication, i.e. $X \subseteq attr(t_1)$, $X \subseteq attr(t_2)$, and $t_1[X] = t_2[X]$.

(A1) FR satisfies $X \xrightarrow{attr} YZ$, so we know that $attr(t_1) \cap (Y \cup Z) = attr(t_2) \cap (Y \cup Z)$. Intersecting with Y at both sides results in $attr(t_1) \cap (Y \cup Z) \cap Y = attr(t_2) \cap (Y \cup Z) \cap Y$. This equation can be reduced to $attr(t_1) \cap Y = attr(t_2) \cap Y$, which is the desired consequence.

(A2) As FR satisfies $X \xrightarrow{attr} Y$, we know that $attr(t_1) \cap Y = attr(t_2) \cap Y$. Due to $X \xrightarrow{attr} Z$, we know that $attr(t_1) \cap Z = attr(t_2) \cap Z$. By building the union of the two equations, we get $(attr(t_1) \cap Y) \cup (attr(t_1) \cap Z) = (attr(t_2) \cap Y) \cup (attr(t_2) \cap Z)$. Applying the distribution law yields $attr(t_1) \cap (Y \cup Z) = attr(t_2) \cap (Y \cup Z)$, hence $X \xrightarrow{attr} YZ$ holds.

(A3) The premise of an attribute dependency assures that $attr(t_1) \cap X = attr(t_2) \cap X$ and therefore, for each subset Y of X, $attr(t_1) \cap Y = attr(t_2) \cap Y$.

(A4) Let t_1, t_2 be two arbitrary tuples of *inst*(FR) with $XZ \subseteq attr(t_1)$, $XZ \subseteq attr(t_2)$, and $t_1[XZ] = t_2[XZ]$. It is obvious that we can infer $X \subseteq attr(t_1)$, $X \subseteq attr(t_2)$, and $t_1[X] = t_2[X]$ from this. As $X \xrightarrow{attr} Y$ holds in FR, and the premise is satisfied, we can conclude that $attr(t_1) \cap Y = attr(t_2) \cap Y$.

Completeness: To start we define X^+_{attr} to be the closure of X (with respect to *AD*) as the set of attributes A such that $X \xrightarrow{attr} A$ can be deduced from the dependency set *AD* by the axioms in $\mathfrak{A}$. Now $X \xrightarrow{attr} Y$ holds if and only if $Y \subseteq X^+_{attr}$. To prove this, we simply note that the addition rule and the projection rule hold for attribute dependencies as well as for functional dependencies, so we can rely upon the analogous proof for functional dependencies (cf. [Ullm88,p.386]).

Now let AD^- be the set of all attribute dependencies that cannot be derived from *AD* by the axioms in $\mathfrak{A}$. To prove the completeness, for each $X \xrightarrow{attr} Y$ in AD^- we have to find a flexible relation FR that satisfies all dependencies in AD^+, but not $X \xrightarrow{attr} Y$. Again we refer to the analogous proof for functional dependencies and construct a two-tuple (flexible) relation with the following specification

$attr(t_1) = \mathfrak{U}$
$\forall A \in \mathfrak{U} : t_1(A) = 1$
$attr(t_2) = X^+_{attr}$
$\forall A \in X : t_2(A) = 1$
$\forall A \in X^+_{attr} - X : t_2(A) = 0$

This relation can be visualized as follows (with //// symbolizing non-existent attributes)

attributes of X	attributes of $X^+_{attr} - X$	attributes of $\mathfrak{U} - X^+_{attr}$
1 1 ... 1	1 1 ... 1	1 1 ... 1
1 1 ... 1	0 0 ... 0	/ / / / / / /

We have to show that $X \xrightarrow{attr} Y$ is not satisfied by this flexible relation. Y cannot be a subset of X^+_{attr}, otherwise it would have been inferred by the closure property. So there is at least one attribute $A \in Y$ lying in $\mathfrak{U} - X^+_{attr}$. By construction t_1 possesses A, but t_2 does not. So $attr(t_1) \cap Y \neq attr(t_2) \cap Y$, although $t_1[X] = t_2[X]$ holds, i.e. $X \xrightarrow{attr} Y$ is not satisfied.
In addition we have to show that FR is a legal relation, i.e. that all dependencies in AD^+ are satisfied. Let $W \xrightarrow{attr} Z \in AD^+$. If $W \not\subseteq X$, then t_1 and t_2 disagree on W, and the dependency is trivially satisfied by FR. Let on the other hand $W \subseteq X$. Then we can apply the left augmentation rule yielding $X \xrightarrow{attr} Z$. The closure property now tells us that $Z \subseteq X^+_{attr}$, and therefore $attr(t_1) \cap Z = attr(t_2) \cap Z$. Hence $W \xrightarrow{attr} Z$ is satisfied by FR. That is, the axioms are complete.

There is still a little problem with this rather close analogy to the completeness proof for functional dependencies. In the conventional relational model there is exactly one scheme corresponding to a given attribute set $\mathcal{U}$. In contrast, in the model of flexible relations any set of subsets of $\mathcal{U}$ is a valid (flexible) scheme. Therefore if we regard a concrete flexible scheme FS the relation constructed above might not be a valid instance of FS.
If we take a concrete flexible scheme FS into account we have to add two further rules to our axiom system. The first rule utilizes the fact that from an unsatisfiable premise we can derive anything. The unsatisfiable premise in our case is a set X of attributes such that no valid attribute set in *dnf*(FS) contains X as a subset.

(A5) $\emptyset$:- $X \xrightarrow{attr} Y$ if $X,Y \subseteq \mathcal{U}, \forall V \in dnf(FS) : X \not\subseteq V$

The second rule will tell us that we cannot disprove a dependency in case of an existential relationship between the attribute sets. Suppose for example that the attributes A_1 and A_2 exclude each other. To disprove $A_1 \xrightarrow{attr} A_2$ we would have to construct two tuples, both possessing A_1 (with the same value) and one of them containing A_2, while the other tuple must not possess A_2, contradicting the exclusion assumption. The same arguments apply if A_1 and A_2 appear only pairwise. This perception results in the rule that the existential implication "the occurence of the attribute set X determines a single subset of Y" is a valid attribute dependency.

Let $W \in dnf(FS)$ such that $X \subseteq W$. Define Z as $Z = W \cap Y$. Then

(A6) $\emptyset$:- $X \xrightarrow{attr} Y$ if $X,Y \subseteq \mathcal{U}, \forall V \in dnf(FS) : X \subseteq V \rightarrow V \cap Y = Z$

The soundness of the two additional rules is informally discussed above and can easily be verified. Note that every attribute dependency stemming from rule (A5) or (A6) can be inferred from a given flexible scheme FS.

Given these two rules we can show the completeness relative to a given flexible scheme FS as follows. Let $X \xrightarrow{attr} Y \in AD^-$. Now we can assume that there are at least two valid attribute sets V_1 and V_2 in *dnf*(FS) such that $X \subseteq V_1$, $X \subseteq V_2$, and $Y \cap V_1 \neq Y \cap V_2$. Otherwise either rule (A5) or (A6) would apply and then $X \xrightarrow{attr} Y \in AD^+$. Now we can construct a flexible relation FR with two tuples t_1 and t_2 as follows.

$attr(t_1) = V_1$
$\forall A \in V_1 : t_1(A) = 1$
$attr(t_2) = V_2$
$\forall A \in X : t_2(A) = 1$
$\forall A \in V_2 - X : t_2(A) = 0$

Now FR is a valid instance of the given scheme FS and, by applying the same arguments as for the simple construction, $X \xrightarrow{attr} Y$ is not satisfied by FR, but any $W \xrightarrow{attr} Z \in AD^+$. That is, the extended axiom system is complete for any given flexible scheme FS.

Non-redundancy: To show the non-redundancy we drop successively one of the four axioms, choose a set AD of attribute dependencies and compute $AD^+_{\text{remaining}}$, the set of all dependencies which can be derived from the remaining three axioms. Then we show that $AD^+_{\text{remaining}}$ does not contain a dependency which can be drawn from the axiom being regarded.

(A1) Let $A, B, C \in \mathcal{U}$. Choose $AD = \{ A \xrightarrow{\text{attr}} BC \}$. $AD^+_{A2,A3,A4}$ can be computed by starting with the initial dependency set AD and by adding dependencies, which are derived by applying any of the rules (A2), (A3) or (A4) to members of the set, until no more dependencies can be derived. We obtain

$$AD^+_{A2,A3,A4} = \{ X \xrightarrow{\text{attr}} Y \mid X \subseteq \mathcal{U}, Y \subseteq X \} \cup \{ AX \xrightarrow{\text{attr}} BCY \mid X \subseteq \mathcal{U}, Y \subseteq AX \}$$

In particular, $A \xrightarrow{\text{attr}} B$ is not contained in $AD^+_{A2,A3,A4}$, but can be derived with rule (A1). Therefore, rule (A1) is not superfluous in $\mathfrak{A}$.

(A2) Let $A, B, C \in \mathcal{U}$. Choose $AD = \{ A \xrightarrow{\text{attr}} B, A \xrightarrow{\text{attr}} C \}$.

$$AD^+_{A1,A3,A4} = \{ X \xrightarrow{\text{attr}} Y \mid X \subseteq \mathcal{U}, Y \subseteq X \} \cup \{ AX \xrightarrow{\text{attr}} B \mid X \subseteq \mathcal{U} \} \cup \{ AX \xrightarrow{\text{attr}} C \mid X \subseteq \mathcal{U} \}$$

One can see that $A \xrightarrow{\text{attr}} BC$ cannot be derived from $AD^+_{A1,A3,A4}$, but from rule (A2). We can conclude that rule (A2) is not implied by the others.

(A3) Without any given dependency ($AD = \emptyset$), nothing can be derived from the rules (A1), (A2) or (A4), i.e. $AD^+_{A1,A2,A4} = \emptyset$. From (A3) we can infer $\{ X \xrightarrow{\text{attr}} Y \mid X \subseteq \mathcal{U}, Y \subseteq X \}$ independent of the given AD, take $\emptyset \xrightarrow{\text{attr}} \emptyset$ as an example. As this attribute dependency is not contained in $AD^+_{A1,A2,A4}$, rule (A3) is non-redundant in $\mathfrak{A}$.

(A4) Let $A, B, C \in \mathcal{U}$. Choose $AD = \{ A \xrightarrow{\text{attr}} B \}$.

$$AD^+_{A1,A2,A3} = \{ X \xrightarrow{\text{attr}} Y \mid X \subseteq \mathcal{U}, Y \subseteq X \} \cup \{ A \xrightarrow{\text{attr}} B, A \xrightarrow{\text{attr}} AB \}$$

In particular, $AC \xrightarrow{\text{attr}} B$ is not contained in $AD^+_{A1,A2,A3}$, but can be deduced from rule (A4). Therefore, rule (A4) is not superfluous in $\mathfrak{A}$.

□

Theorem 4.2 Let $\mathcal{AF}$ be the following system of axioms:

(AF1)	$X \xrightarrow{\text{func}} Y$:- $X \xrightarrow{\text{attr}} Y$	(subsumption)
(AF2)	$\{ X \xrightarrow{\text{func}} Y , Y \xrightarrow{\text{attr}} Z \}$:- $X \xrightarrow{\text{attr}} Z$	(combined transitivity)
(A1)	$X \xrightarrow{\text{attr}} YZ$:- $X \xrightarrow{\text{attr}} Y$	(projectivity)
(A2)	$\{ X \xrightarrow{\text{attr}} Y , X \xrightarrow{\text{attr}} Z \}$:- $X \xrightarrow{\text{attr}} YZ$	(additivity)
(F1)	$\emptyset$:- $X \xrightarrow{\text{func}} Y$ if $Y \subseteq X$	(reflexivity)
(F2)	$X \xrightarrow{\text{func}} Y$:- $XZ \xrightarrow{\text{func}} YZ$	(augmentation)
(F3)	$\{ X \xrightarrow{\text{func}} Y , Y \xrightarrow{\text{func}} Z \}$:- $X \xrightarrow{\text{func}} Z$	(transitivity)

$\mathcal{AF}$ is a sound, complete and non-redundant system of axioms for the implication of functional and attribute dependencies.

Proof.

Soundness: Let FR be a flexible relation. Let t_1, t_2 be two arbitrary tuples of *inst*(FR) with $X \subseteq attr(t_1)$, $X \subseteq attr(t_2)$, and $t_1[X] = t_2[X]$.

The soundness of (A1), (A2), (F1), (F2) and (F3) has already been shown.

(AF1) FR satisfies $X \xrightarrow{\text{func}} Y$ and the premise is given for t_1 and t_2. The consequence of the functional dependency implies $attr(t_1) \cap Y = attr(t_2) \cap Y$, which is the desired result.

(AF2) As FR satisfies $X \xrightarrow{\text{func}} Y$, we know that $Y \subseteq attr(t_1)$, $Y \subseteq attr(t_2)$, and $t_1[Y] = t_2[Y]$. From this and $Y \xrightarrow{\text{attr}} Z$ we can conclude that $attr(t_1) \cap Z = attr(t_2) \cap Z$.

Completeness: In the proof of the completeness of the axiom system $\mathcal{A}$ (see proof of theorem 4.1) we introduced X^+_{attr} , the closure of attribute dependencies for the attribute set X. X^+_{func} , the closure of functional dependencies for X is known from literature [Ullm88,p.386]. The relationship between both closures is that $X^+_{attr} \supseteq X^+_{func}$ as any functional dependency implies an attribute dependency (see the subsumption rule (AF1)) but the converse does not necessarily hold.

To show the completeness we construct again for any $X \longrightarrow Y \in AF^-$ a two-tuple (flexible) relation FR with the following specification (the construction does not depend on if we regard $X \xrightarrow{attr} Y \in AF^-$ or $X \xrightarrow{func} Y \in AF^-$)

$attr(t_1) = \mathcal{U}$
$\forall A \in \mathcal{U} : t_1(A) = 1$
$attr(t_2) = X^+_{attr}$
$\forall A \in X^+_{func} : t_2(A) = 1$
$\forall A \in X^+_{attr} - X^+_{func} : t_2(A) = 0$

This relation can be visualized as follows (with //// symbolizing non-existent attributes)

attributes of X^+_{func}	attributes of $X^+_{attr} - X^+_{func}$	attributes of $\mathcal{U} - X^+_{attr}$
1 1 ... 1	1 1 ... 1	1 1 ... 1
1 1 ... 1	0 0 ... 0	/ / / / / / /

Suppose we have to show that $X \xrightarrow{attr} Y$ is not satisfied by this flexible relation. By reflexivity, $X \subseteq X^+_{func}$, so by construction $t_1[X] = t_2[X]$. Y cannot be a subset of X^+_{attr}, otherwise it would have been inferred by the closure property. Therefore we can find an attribute $A \in Y$ lying in $\mathcal{U} - X^+_{attr}$. By construction t_1 possesses A, but t_2 does not. So $attr(t_1) \cap Y \neq attr(t_2) \cap Y$, i.e. $X \xrightarrow{attr} Y$ is not satisfied.
Suppose at the other hand that we have to show that $X \xrightarrow{func} Y$ is not satisfied by this relation. By the closure property Y cannot be a subset of X^+_{func}, so by construction either $Y \not\subseteq attr(t_2)$ or at least $t_1[Y] \neq t_2[Y]$. In any case $X \xrightarrow{func} Y$ is not satisfied[11].
In addition we have to show that FR is a legal relation, i.e. that all dependencies in AF^+ are satisfied. Let $W \xrightarrow{func} Z \in AF^+$. If $W \not\subseteq X^+_{func}$, then t_1 and t_2 disagree on W, and the dependency is trivially satisfied by FR. Let on the other hand $W \subseteq X^+_{func}$. Then by the closure property $X \xrightarrow{func} W$ and by transitivity $X \xrightarrow{func} Z$. Using the closure property again we get $Z \subseteq X^+_{func}$ and now, by construction, $t_1[Z] = t_2[Z]$. Hence $W \xrightarrow{func} Z$ is satisfied by FR.
Take now $W \xrightarrow{attr} Z \in AF^+$. Again, if $W \not\subseteq X^+_{func}$, then the dependency is trivially satisfied by FR. Assume on the other hand $W \subseteq X^+_{func}$. From the functional closure property we can infer that $X \xrightarrow{func} W$. Now the combined transitivity applies[12] and yields that $X \xrightarrow{attr} Z$ holds. The attribute closure property asserts that $Z \subseteq X^+_{attr}$ and, by construction, $attr(t_1) \cap Z = attr(t_2) \cap Z$. Hence $W \xrightarrow{attr} Z$ is satisfied by FR.

The extension to take a concrete flexible scheme into account is a direct analogy to the proof of Theorem 4.1 and is therefore omitted.

---

[11] Note that when $Y \subseteq X^+_{attr}$ and $Y \not\subseteq X^+_{func}$ then $X \xrightarrow{attr} Y$ still holds although $X \xrightarrow{func} Y$ does not.

[12] Note that every rule of the axiom system has been employed now in this proof, validating again the ingenuity of the rule system.

Non-redundancy: To show the non-redundancy we drop successively one of the seven axioms, choose a set AF of attribute and functional dependencies and compute $AF^+_{\text{remaining}}$, the set of all dependencies which can be derived from the remaining six axioms. Then we show that $AF^+_{\text{remaining}}$ does not contain a dependency which can be drawn from the axiom being regarded.

The non-redundancy of (F1), (F2) and (F3) is obvious as, taken alone they are non-redundant (cf. [PDGV89,p.67f]), and none of the other rules have functional dependencies as their consequence.

(A1) Let $A, B, C \in \mathcal{U}$. Choose $AF = \{ A \xrightarrow{\text{attr}} BC \}$. $AF^+_{A2,F1,F2,F3,AF1,AF2}$ can be computed by starting with the initial dependency set AF and by adding dependencies, which are derived by applying any of the rules (A2), (F1), (F2), (F3), (AF1) or (AF2) to members of the set, until no more dependencies can be derived. We obtain

$$AF^+_{A2,F1,F2,F3,AF1,AF2} = \{ X \xrightarrow{\text{func}} Y \mid X \subseteq \mathcal{U}, Y \subseteq X \} \cup \{ X \xrightarrow{\text{attr}} Y \mid X \subseteq \mathcal{U}, Y \subseteq X \} \cup \{ AX \xrightarrow{\text{attr}} BCY \mid X \subseteq \mathcal{U}, Y \subseteq AX \}$$

In particular, $A \xrightarrow{\text{attr}} B$ is not contained in $AF^+_{A2,F1,F2,F3,AF1,AF2}$, but can be derived with rule (A1). Therefore, rule (A1) is not superfluous in $\mathcal{AF}$.

(A2) Let $A, B, C \in \mathcal{U}$. Choose $AF = \{ A \xrightarrow{\text{attr}} B, A \xrightarrow{\text{attr}} C \}$.

$$AF^+_{A1,F1,F2,F3,AF1,AF2} = \{ X \xrightarrow{\text{func}} Y \mid X \subseteq \mathcal{U}, Y \subseteq X \} \cup \{ X \xrightarrow{\text{attr}} Y \mid X \subseteq \mathcal{U}, Y \subseteq X \} \cup \{ AX \xrightarrow{\text{attr}} B \mid X \subseteq \mathcal{U} \} \cup \{ AX \xrightarrow{\text{attr}} C \mid X \subseteq \mathcal{U} \}$$

One can see that $A \xrightarrow{\text{attr}} BC$ cannot be derived from $AF^+_{A1,F1,F2,F3,AF1,AF2}$, but from rule (A2). We can conclude that rule (A2) is not implied by the others.

(AF1) Let $A, B \in \mathcal{U}$. Choose $AF = \{ A \xrightarrow{\text{func}} B \}$.

$$AF^+_{A1,A2,F1,F2,F3,AF2} = \{ X \xrightarrow{\text{func}} Y \mid X \subseteq \mathcal{U}, Y \subseteq X \} \cup \{ AX \xrightarrow{\text{func}} BY \mid X \subseteq \mathcal{U}, Y \subseteq X \} \cup \{ AX \xrightarrow{\text{func}} ABY \mid X \subseteq \mathcal{U}, Y \subseteq X \}$$

From (AF1) we can infer $A \xrightarrow{\text{attr}} B$. As this attribute dependency is not contained in $AF^+_{A1,A2,F1,F2,F3,AF2}$, rule (AF1) is non-redundant in $\mathcal{AF}$.

(AF2) Let $A, B, C \in \mathcal{U}$. Choose $AF = \{ A \xrightarrow{\text{func}} B, B \xrightarrow{\text{attr}} C \}$.

$$AF^+_{A1,A2,F1,F2,F3,AF1} = \{ X \xrightarrow{\text{func}} Y \mid X \subseteq \mathcal{U}, Y \subseteq X \} \cup \{ X \xrightarrow{\text{attr}} Y \mid X \subseteq \mathcal{U}, Y \subseteq X \} \cup \{ AX \xrightarrow{\text{func}} BY \mid X \subseteq \mathcal{U}, Y \subseteq AX \} \cup \{ AX \xrightarrow{\text{attr}} BY \mid X \subseteq \mathcal{U}, Y \subseteq AX \} \cup \{ B \xrightarrow{\text{attr}} X \mid X \subseteq BC \}$$

In particular, $A \xrightarrow{\text{attr}} C$ is not contained in $AF^+_{A1,A2,F1,F2,F3,AF1}$, but can be deduced from rule (AF2). Therefore, rule (AF2) is not superfluous in $\mathcal{AF}$.

□

Theorem 4.3 Let *ads*(FR) be the set of attribute dependencies that hold in the flexible relation FR. The following rules describe the propagation of attribute dependencies:

(1) $ads(FR_1 \times FR_2) = ads(FR_1) \cup ads(FR_2)$

(2) $ads(\pi_X(FR)) = \{ V \xrightarrow{attr} W \cap X \mid V \xrightarrow{attr} W \in ads(FR) \wedge V \subseteq X \}$

(3) $ads(\sigma_F(FR)) = ads(FR)$

(4) $ads(FR_1 \cup FR_2) = \varnothing$

(5) $ads(FR_1 - FR_2) = ads(FR_1)$

(6) $ads((\varepsilon_{A:a1}(FR_1)) \cup (\varepsilon_{A:a2}(FR_2))) =$
$\{ AX \xrightarrow{attr} Y \mid X \xrightarrow{attr} Y \in ads(FR_1) \wedge X \xrightarrow{attr} Y \in ads(FR_2) \}$

(7) $ads((\varepsilon_{A1:a1}(\varepsilon_{A2:unique()}(FR_1))) \cup (\varepsilon_{A1:unique()}(\varepsilon_{A2:a2}(FR_2)))) =$
$\{ A_1X \xrightarrow{attr} Y \mid X \xrightarrow{attr} Y \in ads(FR_1) \} \cup \{ A_2X \xrightarrow{attr} Y \mid X \xrightarrow{attr} Y \in ads(FR_2) \}$

Proof.

(1) As usual we demand for the cartesian product that $attr(FR_1) \cap attr(FR_2) = \varnothing$ and that $\pi_{attr(FR_1)}(FR_1 \times FR_2) = FR_1$. W.l.o.g. let $X \xrightarrow{attr} Y \in ads(FR_1)$.
Let $t_1,t_2 \in inst(FR_1 \times FR_2)$ fulfill the premise of $X \xrightarrow{attr} Y$, i.e. $X \subseteq attr(t_1) \wedge X \subseteq attr(t_2) \wedge t_1[X] = t_2[X]$. Let $t_1' = t_1[attr(FR_1)]$ and $t_2' = t_2[attr(FR_1)]$.
As $X \subseteq attr(FR_1)$ we derive that $X \subseteq attr(t_1') \wedge X \subseteq attr(t_2') \wedge t_1'[X] = t_2'[X]$. Due to $\pi_{attr(FR_1)}(FR_1 \times FR_2) = FR_1$ we know that $t_1',t_2' \in inst(FR_1)$ and therefore the consequence of $X \xrightarrow{attr} Y$ holds, i.e. $attr(t_1') \cap Y = attr(t_2') \cap Y$. As $Y \subseteq attr(FR_1)$ we conclude that $attr(t_1) \cap Y = attr(t_2) \cap Y$, i.e. $X \xrightarrow{attr} Y$ holds in $FR_1 \times FR_2$.

(2) Let $V \xrightarrow{attr} W \in ads(FR)$ and $V \not\subseteq X$. Now we can always construct two tuples t_1,t_2 such that $t_1[V] \neq t_2[V]$ but $t_1[X] = t_2[X]$, i.e. their projections onto X coincide. But as the tuples disagreed on V we must not conclude a dependency although $t_1[X] = t_2[X]$ and therefore $V \xrightarrow{attr} W$ cannot be preserved in the projection.
Suppose on the other hand that $V \subseteq X$. Now $t_1[V] \neq t_2[V]$ implies $t_1[X] \neq t_2[X]$, i.e. no "false drops" may be generated. Considering that only the attributes inside X are preserved we conclude that $V \xrightarrow{attr} W \cap X$ holds in $\pi_X(FR)$.

(3) As $inst(\sigma_F(FR)) \subseteq inst(FR)$, the result is obvious[13].

---

[13] Note that if one treats the selection as potentially type modifying like our algebra does (consider $\sigma_{\text{job-type = 'salesman'}}$ (employee)), then some of the attribute dependencies may become trivial. Nevertheless, they cannot become wrong.

(4) No attribute dependency may hold in a union as one cannot decide from which input relation the tuples do come from. An attribute dependency that holds in one of the input relations may always be violated by tuples of the other relation. Now suppose that $X \xrightarrow{attr} Y$ holds in both input relations. Let $t_{11},t_{12} \in inst(FR_1)$ and $t_{21},t_{22} \in inst(FR_2)$ and $t_{11}[X] = t_{12}[X] = t_{21}[X] = t_{22}[X]$. Let further $attr(t_{11}) \cap Y = attr(t_{12}) \cap Y \neq attr(t_{21}) \cap Y = attr(t_{22}) \cap Y$. The construction is valid, i.e. $X \xrightarrow{attr} Y$ is satisfied in both input relations, but it does not hold in the union. We conclude that no attribute dependency holds in a union.

(5) As $inst(FR_1 - FR_2) \subseteq inst(FR_1)$, the result is obvious. Attribute dependencies that hold in FR_2 must not be omitted due to the non-monotonicity of the minus operator.

(6) Let $t_1,t_2 \in inst((\varepsilon_{A:a1} (FR_1)) \cup (\varepsilon_{A:a2} (FR_2)))$ fulfill the premise of $AX \xrightarrow{attr} Y$, i.e. $AX \subseteq attr(t_1) \wedge AX \subseteq attr(t_2) \wedge t_1[AX] = t_2[AX]$. Let $t_1' = t_1[attr(FR_1)]$ and $t_2' = t_2[attr(FR_1)]$.
W.l.o.g. suppose that $t_1(A) = t_2(A) = a_1$ and $a_1 \neq a_2$, from which we may derive that $t_1',t_2' \in inst(FR_1)$. As $X \xrightarrow{attr} Y \in ads(FR_1)$ and t_1',t_2' fulfill its premise, we know that $attr(t_1') \cap Y = attr(t_2') \cap Y$. As $A \notin Y$ we conclude that $attr(t_1) \cap Y = attr(t_2) \cap Y$, i.e. $AX \xrightarrow{attr} Y$ holds in $(\varepsilon_{A:a1} (FR_1)) \cup (\varepsilon_{A:a2} (FR_2))$.

(7) Let $t_1,t_2 \in inst((\varepsilon_{A1:a1} (\varepsilon_{A2:unique()} (FR_1))) \cup (\varepsilon_{A1:unique()} (\varepsilon_{A2:a2} (FR_2))))$ fulfill the premise of $A_1X \xrightarrow{attr} Y$, i.e. $A_1X \subseteq attr(t_1) \wedge A_1X \subseteq attr(t_2) \wedge t_1[A_1X] = t_2[A_1X]$. Let $t_1' = t_1[attr(FR_1)]$ and $t_2' = t_2[attr(FR_1)]$. As each tuple of FR_2 was extended with a unique value in attribute A_1 we know that $t_1',t_2' \in inst(FR_1)$. As $X \xrightarrow{attr} Y \in ads(FR_1)$ and t_1',t_2' fulfill its premise, we know that $attr(t_1') \cap Y = attr(t_2') \cap Y$. As $A_1,A_2 \notin Y$ we conclude that $attr(t_1) \cap Y = attr(t_2) \cap Y$, i.e. $A_1X \xrightarrow{attr} Y$ holds in $(\varepsilon_{A1:a1} (\varepsilon_{A2:unique()} (FR_1))) \cup (\varepsilon_{A1:unique()} (\varepsilon_{A2:a2} (FR_2)))$. The same arguments apply to the attribute dependencies of FR_2.

□