Semantics for Null Extended Nested Relations

MARK LEVENE

University College, University of London

and

GEORGE LOIZOU

Birkbeck College, University of London†

The nested relational model extends the flat relational model by relaxing the first normal form assumption in order to allow the modelling of complex objects. Much of the previous work on the nested relational model has concentrated on defining the data structures and query language for the model. The work done on integrity constraints in nested relations has mainly focused on characterising subclasses of nested relations and defining normal forms for nested relations with certain desirable properties.

In this paper we define the semantics of nested relations, which may contain null values, in terms of integrity constraints, called *null extended data dependencies*, which extend functional dependencies and join dependencies encountered in flat relational database theory. We formalise incomplete information in nested relations by allowing only one unmarked *generic null value*, whose semantics we do not further specify. The motivation for the choice of a generic null is our desire to investigate only fundamental semantics which are common to all unmarked null types. This leads us to define a preorder on nested relations, which allows us to measure the relative information content of nested relations. We also define a procedure, called the *extended chase procedure*, for testing satisfaction of null extended data dependencies and for making inferences by using these null extended data dependencies. The extended chase procedure is shown to

† Authors' full addresses: M. Levene, Department of Computer Science, University College London, Gower Street, London WC1E 6BT, U.K., E-Mail address: M.Levene@uk.ac.ucl.cs; G. Loizou, Department of Computer Science, Birkbeck College, University of London, Malet Street, London WC1E 7HX, U.K. This paper appears in *ACM Transactions on Database Systems*, Vol. 18, pp. 414-459.

generalise the classical chase procedure, which is of major importance in flat relational database theory. As a consequence of our approach we are able to capture the novel notion of losslessness in nested relations, called herein *null extended lossless decomposition*. Finally, we show that the semantics of nested relations are a natural extension of the semantics of flat relations.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design-*data models*, *normal forms*; H.2.3 [**Database Management**]: Languages-*query languages*

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: Nested relations, nulls, null extended algebra, null extended data dependencies, extended chase

1. INTRODUCTION

The *nested relational model* [ABIT86, ABIT89, GYSS89, LEVE90b, MAKI77, OZSO87, ROTH88, ROTH89, SCHE86, THOM86, VANG88] was developed in order to extend the applicability of the (flat) relational model [CODD79, MAIE83, ULLM88] to more complex, non-business applications such as CAD, image processing and text retrieval [ABIT89]. Nested relations can model hierarchical complex data directly by recursively defining values of attribute domains to be either atomic or relation-valued. Thus, the nested relational model relaxes the first normal form assumption, which provides the basis for the flat relational model [CODD79].

A database model consists of three main components, which we now enumerate:

(1) The data structures of the model (nested relations).

(2) The query (and update) language of the model (the extended algebra).

(3) The integrity constraints of the model (extended data dependencies).

So far most of the research on the nested relational model has concentrated on (1) and (2) while (3) has been mainly investigated in order to characterise subclasses of nested relations [ABIT86, BIDO87, FISC85, JAES82, MAKI77, MIUR86, ROTH88, ROTH89, TAKE89, THOM86, VANG88] and in order to define normal forms for nested relations [LEVE89b, MAKI77, OZSO87, OZSO89, ROTH88]. Furthermore, there have been only few extensions of the nested relational model which comprehensively incorporate nulls into the model [LEVE89a, LEVE90b, ROTH89]. The treatment of incomplete information [CODD86, IMIE84, LERA86, LEVE89a, MAIE83, ZANI84] within the nested relational model is indeed an important issue, since, in general, we do not expect to have a complete model of the real world. In this respect, we need only mention the controversial issue of giving semantics to relation-valued attributes of nested relations whose value may be the empty set [ABIT86, GYSS89, LEVE89a, LEVE90a, MAKI77, ROTH89].

In the present paper we define the semantics of nested relations in terms of:

(1) the *relative information content* of tuples in *null extended nested relations* (which we will refer to from now on simply as nested relations);

(2) the class of integrity constraints, called *null extended data dependencies*, that hold in nested relations;

(3) the *extended chase procedure* (or simply the extended chase), which allows us to test the satisfaction of a given set of null extended data dependencies, say D*, and to infer more information from a given nested relation by using D*.

The class of null extended data dependencies includes the following three types of integrity constraint. Firstly, *null functional dependencies* (NFDs) which redefine NFDs that hold in flat relations (cf. [ATZE86, LIEN82]) within the context of nested relations. NFDs in turn generalise *functional dependencies* (FDs) [MAIE83, ULLM88] so as to allow unmarked null values. Secondly, *null extended functional dependencies* (NEFDs) (cf. [FISC85, JAES82, MAKI77, MIUR86, ROTH88, ROTH89, THOM86, VANG88]) which extend NFDs from flat relations to nested relations in a natural way by having relation-valued attributes on the right-hand side. Lastly, *null extended join dependencies* (NEJDs) which extend *join dependencies* (JDs) [BERR81, SCIO82] and thus also the fundamental notion of a *lossless join decomposition* [MAIE83, ULLM88] to nested relations. As a consequence of this last null extended data dependency, we are able to capture the novel notion of losslessness in nested relations, which we call *null extended lossless decomposition*.

We show that null extended data dependencies that hold in nested relations have counterparts in the class of *null data dependencies* that hold in the flat relations emanating from the said nested relations. More specifically, the counterpart of a NFD holding in a nested relation is a NFD, the counterpart of a NEFD is a *null extended multivalued dependency* (NEMVD) (cf. [LIEN79, LIEN82]) and the counterpart of a NEJD is a *null join dependency* (NJD) (cf. [JAJO90]), which generalises JDs, defined for flat relations without nulls, to null extended flat relations (or simply flat relations). Furthermore, as in [FAGI82] we conjecture that the class of null extended data dependencies considered herein is sufficient to model most real-world applications.

Example 1.1. Schemas of nested relations are represented by *scheme trees* [OZSO87, OZSO89], as shown in Figure 1.1. The *nested relation scheme* (NRS) for the scheme tree, T, denoted by R(T), is: TUTOR SALARY (CHILD)* (DAY)*, where attributes with relation-valued domains are marked by * in order to distinguish them from attributes defined over atomic domains [ABIT86, OZSO87].

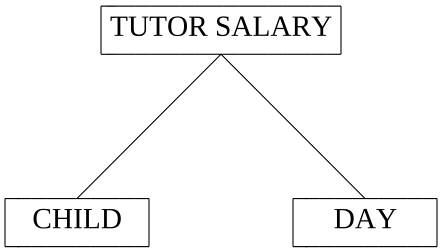

Fig. 1.1. The scheme tree T.

A nested relation, r*, over the NRS, R(T) = TUTOR SALARY (CHILD)* (DAY)*, for the scheme tree, T, of Figure 1.1, is shown in Figure 1.2. We note that *null* in r* denotes a null value therein. In this paper we consider *null* to be a *generic null value*, i.e. we do not further specify its semantics, apart from the fact that *null* is *less informative* than any non-null value, v, i.e. *null* ≤ v.

The motivation for the choice of a generic null is our desire to investigate only the fundamental semantics which are common to all unmarked null types. For example, for the third tuple in r*, we could interpret *null* over (CHILD)* to mean that Martine does not have any children, or we could interpret this *null* to mean that Martine has one or more children whose names are unknown.

The semantics of the nested relation, r*, over the NRS, R(T), can be given by the following set of null extended data dependencies, namely, the NFD: TUTOR → SALARY, and the NEFDs: TUTOR,SALARY → (CHILD)* and TUTOR,SALARY → (DAY)*. In other words, a TUTOR, who has a unique SALARY, has a unique set of CHILDren and gives tutorials on a unique set of DAYs.

TUTOR	SALARY	(CHILD)*	(DAY)*
		CHILD	DAY
Robert	12000	Hanna Brian	Monday Thursday
Hanna	14000	Annette Ada	*null*
Martine	*null*	*null*	*null*
null	15000	*null*	Wednesday
null	*null*	Ruth	Tuesday Friday

Fig. 1.2. The nested relation, r*, over the nested relation scheme, R(T).

The *chase procedure* (or simply the chase) [ATZE87, GRAH86, HONE82, MAIE79, MAIE84, MEND84, SAGI83, SAGI88, STEI85] is of major significance in flat relational database theory providing a means of testing the satisfaction of a set of data dependencies, D, and of making inferences from a flat database by using D. We show that the extended chase with respect to a set of null extended data dependencies, D*, applied to a nested relation, r*, has the same information content as a chase with respect to the counterpart set of null data dependencies of D* applied to the flat relation emanating from r*.

Thus, the semantics we define in this paper for the nested relational model in the presence of nulls is a natural extension of the standard semantics given to the flat relational model in the sense that the flat relational model is a special case of the nested relational model. In addition, the semantics of a nested relation always has counterpart semantics in the corresponding flat relation.

The rest of the paper is organised as follows. In Section 2 we define nested relation schemes and null extended nested relations. In Section 3 we define the operators of the *null extended algebra* necessary for the formalism of null extended data dependencies, which are defined in Section 4. In Section 5 we extend the chase to nested relations and, finally, in Section 6 we give our conclusions.

2. NULL EXTENDED NESTED RELATIONS AND THEIR SCHEMES

In this section we define the data structures of the null extended nested relational model. In Subsection 2.1 we define nested relation schemes and in Subsection 2.2 we define null extended nested relations (or simply nested relations), which may include generic *nulls*. The generic null type, *null*, presented in this paper is not given specific semantics such as: *value unknown* [CODD79, CODD86, LIPS79] or *value does not exist* [CODD86, LERA86], since herein we are only interested in the semantics common to all unmarked null types. We can then define the relative information content of tuples and nested relations by generalising the *Hoare ordering* on powerdomains [SCHM86] to nested relations (cf. [BUNE91, LEVE90b]). This preorder allows us to define *information-wise equivalent* nested relations and a canonical representation of nested relations by using the concept of *reduction* [LEVE89a, ROTH89, ZANI84]. Additionally, in our semantics of (unmarked) null values we define the *inequality rule for nulls* which states that *null* ≠ *null*. The justification for this inequality rule is that *null* is defined to be *less informative* than any non-null value and thus when two *nulls* are updated they may be replaced by two distinct non-null values.

2.1. Nested Relation Schemes

In this subsection we define scheme trees and nested relation schemes.

Definition 2.1. Let U = $\{A_1, A_2, ..., A_p\}$ be the universal set of attributes and let W $\subseteq$ U. Then, a *scheme tree*, T, defined over the set of attributes, W, is a rooted tree whose vertices are labelled by pairwise disjoint subsets of W.

The following functions which operate on a scheme tree, T, are now defined.

(1) ATT(n) is a label for node, n, and is equal to the set of attributes labelling the node n;

(2) A(n) is the union of all ATT(v) for all ancestor nodes v of n, including ATT(n);

(3) D(n) is the union of all ATT(v) for all descendant nodes v of n, including ATT(n);

(4) S(T) is the union of all ATT(n) for all nodes in T;

(5) ROOT(T) returns the root node of T.

A *scheme forest*, F, over U, is a collection $\{T_1,T_2,...,T_q\}$ of scheme trees such that S(T_i) $\subseteq$ U, $1 \leq i \leq q$, and S(F) = $\cup_{i=1}^{q}$ S(T_i) = U.

If $u_1,u_2,...,u_m$ are all the leaf nodes of T, then the *path set* of T, P(T), is given by P(T) = {A(u_1),A(u_2),...,A(u_n)}.

Following Abiteboul and Bidoit [ABIT86] and Ozsoyoglu and Yuan [OZSO87], we next define the nested relation scheme represented by a scheme tree, T.

Definition 2.2. The *nested relation scheme* (NRS), represented by a scheme tree, T, denoted by R(T), is defined recursively as a set of attributes by:

(1) if the scheme tree, T, is empty, i.e. T is defined over the attribute set $\emptyset \subseteq$ U, then R(T) = Λ (i.e. we denote the empty set of attributes by the empty string Λ);

(2) if the scheme tree, T, comprises a single node, n, and ATT(n) = X, then R(T) = X;

(3) if X = ATT(ROOT(T)) and T_1, T_2, ..., T_s, $s \geq 1$, denote the first level subtrees of the scheme tree, T, with corresponding

attributes $(R(T_1))^*$, $(R(T_2))^*$, ..., $(R(T_s))^*$, then $R(T) = X \cup \{(R(T_1))^*, (R(T_2))^*, \ldots, (R(T_s))^*\}$ (i.e. we denote the attributes associated with NRSs $R(T_i)$, $1 \leq i \leq s$, by $(R(T_i))^*$).

For notational convenience we also represent the NRS R(T) by the string $X(R(T_1))^*(R(T_2))^*\ldots(R(T_s))^*$. The empty string, Λ, is retained in the substring, $(R(T_1))^*(R(T_2))^*\ldots(R(T_s))^*$, only when it is associated with the root of a tree (or subtree) which itself has at least one subtree which is not empty. In analogy to the standard notation, by $Y \subseteq R(T)$ we mean a substring of R(T) composed of not necessarily consecutive elements, for example, $Y = X'(R(T_2))^*(R(T_s))^*$, with $X' \subseteq X$.

We denote a NRS, R(T), where S(T) = U, by U(T). Furthermore, we let $Z(R(T)) = R(T) \cap U$ be the set of attributes in R(T) associated with atomic domains; such attributes are called the *zero order* attributes of R(T). Correspondingly, we let H(R(T)) = R(T) - Z(R(T)) be the set of attributes in R(T) associated with relation-valued domains; such attributes are called the *higher order* attributes of R(T).

We observe that the notation for higher order attributes using ()* is convenient in our formalism since it highlights their internal structure. A more user-friendly notation would be to give each higher order attribute, $(R(T_i))^*$, a *higher order name* determined by the user as it is done in the nested relational formalisms found in [LEVE90b, ROTH89, SCHE86, THOM86].

Example 2.1. Let T_1 be the scheme tree over W = {STUDENT, DEPT, MAJOR, CLASS, EXAM, PROJECT}, shown in Figure 2.1. Thus, we have $S(T_1)$ = W and $P(T_1)$ = {{STUDENT, DEPT, MAJOR}, {STUDENT, DEPT, CLASS, EXAM}, {STUDENT, DEPT, CLASS, PROJECT}}. Moreover, we have the NRS, $R(T_1)$ = STUDENT DEPT (MAJOR)* (CLASS (EXAM)* (PROJECT)*)*.

A *flat relation scheme* (FRS) is a special case of a NRS, when $R(T) = Z(R(T)) \subseteq U$, i.e. when R(T) includes only zero order attributes.

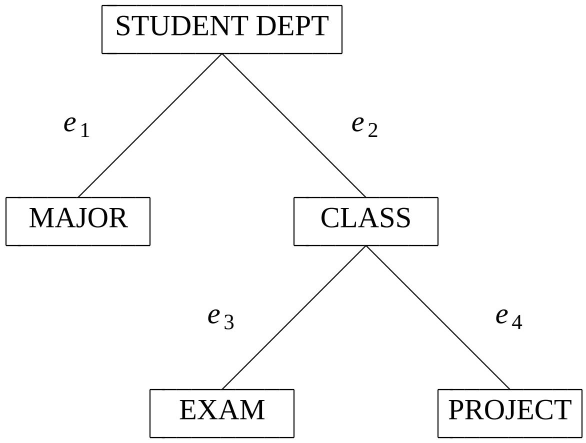

Fig. 2.1. The scheme tree T_1.

A *nested database scheme* (NDS) **R**(F), over U, is a collection {R(T_1),R(T_2),...,R(T_q)} of NRSs such that F = {T_1,T_2,...,T_q} is a scheme forest over U. A *flat database scheme* (FDS) **R**, over U, is a collection {R_1, R_2, ..., R_q} of FRSs, i.e. it is a special case of a NDS. The FDS induced by F, denoted as FDS(F), is given by {S(T_1),S(T_2),...,S(T_q)}. We note that P(T) is a FDS over S(T).

2.2. Null Extended Nested Relations

In this subsection we define null extended nested relations. We begin by constructing the underlying powerdomain whose elements are nested relations. The construction is realised via a set **D** of countable flat domains [SCHM86] each such domain comprising atomic values and a generic unmarked null, denoted by *null*.

Let D $\in$ **D** be a countable flat domain comprising atomic values and a bottom element $\perp$. That is $\forall\, v_i, v_j \in$ D $v_i \leq v_j$ if and only if $v_i = v_j$ or $v_i = \perp$ [SCHM86]. We observe that the partial order $\leq$ on D induces a *star graph* [HARA69].

We interpret the bottom element as being a generic unmarked null value, i.e. *null*, in the sense that *null* ($\perp$) contains less information than any other value in D. Thus, in our formalism we consider only one null value, *null*, which will be a member of all the available domains. We observe that we can extend our formalism to include several unmarked null types in our domains, such as: *value unknown* [CODD79, CODD86, LIPS79], *value does not exist* [CODD86, LERA86] and *no information* [ROTH89, ZANI84], by considering a *semi-lattice domain* (or *tree domain* [BUNE91]) of values [SCHM86]. We justify our choice of a generic null by our desire to investigate only the fundamental semantics which are common to all unmarked null types.

Let **D** = $\{D_1, D_2, ..., D_p\}$ be a set of countable flat domains each consisting of atomic values together with *null*. We associate with each attribute A_i of the universe U a domain, denoted by DOM(A_i) and given by D_i.

Definition 2.3. We define the domain of a NRS R(T), denoted as DOM(R(T)), recursively by:

(1) if the scheme tree T is empty, i.e. R(T) = Λ, then DOM(R(T)) = ∅;

(2) if the scheme tree T comprises a single node n and ATT(n) = X, where X = $\{A_1, A_2, ..., A_m\} \subseteq$ U, m ≥ 1 and m ≤ p, then

DOM(X) = {*null*} + (DOM(A_1) × DOM(A_2) × ... × DOM(A_m))

where + is the disjoint union operator and × is the Cartesian product operator;

(3) let X = ATT(ROOT(R(T))), let $T_1, T_2, ..., T_s$, s ≥ 1, denote the first level subtrees of the scheme tree T and let

DOM($T_1, T_2, ..., T_s$) = *P*(DOM(R(T_1))) × *P*(DOM(R(T_2))) × ... × *P*(DOM(R(T_s))),

where *P* stands for the finite powerset operator. Then,

if X ≠ Λ, DOM(R(T)) = {*null*} + (DOM(X) × DOM($T_1, T_2, ..., T_s$)),

otherwise (X = Λ) DOM(R(T)) = {*null*} + DOM($T_1, T_2, ..., T_s$).

We now define a *null extended nested relation* (abbreviated to nested relation), r*, over a NRS R(T), as an element of *P*(DOM(R(T))), i.e. a finite set of tuples over R(T). If r* = {*null*} then we consider r* to be undefined. A *null extended flat relation* (abbreviated to flat relation) is a special case of a nested relation, i.e. when R(T) = Z(R(T)) is a set of attributes R ⊆ U, that is

R(T) is a FRS.

Example 2.2. A nested relation, r_1^*, over R(T_1) of Example 2.1, is shown in Figure 2.2.

STUDENT	DEPT	(MAJOR)*	(CLASS	(EXAM)*	(PROJECT)*)*
		MAJOR	CLASS	(EXAM)*	(PROJECT)*
				EXAM	PROJECT
Iris	CS	computing	databases	mid final	1NF
			programming	final	*null*
Mark	CS	maths	databases	final	NF2 UR
David	philosophy	logic	first-order	mid	prolog
null	philosophy	*null*	*null*	mid final	*null*
			first-order	*null*	functions predicates
Naomi	*null*	languages	french	mid	*null*
			null	final	Moscow
			hebrew	*null*	Genesis

Fig. 2.2. The nested relation r_1^* over R(T_1).

We define *projection* of a tuple t ∈ r* onto Y ⊆ R(T), denoted by t[Y], to be the restriction of t to Y. We also refer to t[Y] as the Y-value of t. We extend the definition of projection to r* as follows:

r*[Y] = {t[Y] | t ∈ r*}.

A *null extended nested database* (abbreviated to a nested database), d*, over a NDS, **R**(F), is defined by d* = $\{r_1^*, r_2^*, ..., r_q^*\}$, where each $r*_i$ is a nested relation over R(T_i), $1 \leq i \leq q$. A *null extended flat database* (abbreviated to flat database) is a special case of a nested database, i.e. when each R(T_i) = Z(R(T_i)), that is a set of attributes $R_i \subseteq U$, $1 \leq i \leq q$, and thus **R**(F) is the FDS **R** = $\{R_1, R_2, ..., R_q\}$.

Next we extend ≤ to be a preorder (i.e. a reflexive and transitive "relation") on tuples over a NRS, R(T), thus generalising the *Hoare ordering* [SCHM86].

Definition 2.4. Let R(T) = X(R(T_1))*(R(T_2))*...(R(T_s))*, then

(1) *null* $\leq$ t and t $\leq$ t, for any tuple, t, over R(T);

(2) let t_1 and t_2 be two tuples over R(T), then $t_1 \leq t_2$ if and only if

(2.1) $t_1[A_i] \leq t_2[A_i] \;\; \forall A_i \in$ X; and

(2.2) $\forall$ i $\in$ {1,2,...,s} $t_1[(R(T_i))^*] \sqsubseteq t_2[(R(T_i))^*]$, where $\sqsubseteq$ denotes the Hoare ordering on nested relations, i.e. $\forall\, w_1 \in r_1^* \;\exists\, w_2 \in r_2^*$ such that $w_1 \leq w_2$, where r_1^* is the nested relation $t_1[(R(T_i))^*]$ over R(T_i) and r_2^* is the nested relation $t_2[(R(T_i))^*]$ over R(T_i).

We say that t_1 is *less informative* than t_2 if $t_1 \leq t_2$; correspondingly, we say that t_2 is *more informative* than t_1. For nested relations r_1^* and r_2^*, over a NRS R(T), we say that r_1^* is *less informative* than r_2^* if $r_1^* \sqsubseteq r_2^*$; correspondingly, we say that r_2^* is *more informative* than r_1^*.

We say that a tuple t_1 over a NRS R(T) is *information-wise equivalent* to a tuple t_2 over R(T), denoted by $t_1 \cong t_2$, if and only if $t_1 \leq t_2$ and $t_2 \leq t_1$. Correspondingly, we say that a nested relation r_1^* over R(T) is *information-wise equivalent* to a nested relation r_2^* over R(T), also denoted by $r_1^* \cong r_2^*$, if and only if $r_1^* \sqsubseteq r_2^*$ and $r_2^* \sqsubseteq r_1^*$.

For the rest of the paper we do not distinguish between members in each of the equivalence classes of $\cong$ with respect to either tuples or nested relations. The justification for this approach is that we consider the information content of all tuples and all nested relations in an equivalence class to be the same.

A minimal element in each of the equivalence classes of $\cong$ with respect to nested relations can be obtained by the process of *reduction*, which we now define. A *reduced nested relation* is a nested relation from which all *less informative* tuples have been removed. If r* is a nested relation over R(T) then the corresponding *reduced nested relation* is denoted by $\hat{\{}$ r* $\hat{\}}$ [ROTH89]. The reduction of r* is, in general, advantageous since redundancy is removed and thus we get a compact representation of the relative information content of a nested relation containing *nulls*.

Example 2.3. Let r* be the nested relation shown in Figure 2.3; r* is not reduced, since $<b_1, null> \leq <b_1, c_1>$ and $<null, \{<b_1, c_1>\}> \leq <a_1, \{<b_1, c_1>\}>$ both hold. The reduction of r*, $\hat{\{}\, r^* \,\hat{\}}$, is shown in Figure 2.4. It is easy to verify that $r^* \sqsubseteq \hat{\{}\, r^* \,\hat{\}}$ and $\hat{\{}\, r^* \,\hat{\}} \sqsubseteq r^*$ both hold, and thus $r^* \cong \hat{\{}\, r^* \,\hat{\}}$ as required.

A	(B	C)*
	B	C
a_1	b_1	c_1
	b_1	*null*
null	b_1	c_1
null	b_2	c_1

Fig. 2.3. The non-reduced nested relation, r*.

A	(B	C)*
	B	C
a_1	b_1	c_1
null	b_2	c_1

Fig. 2.4. The reduced nested relation, $\hat{\{}\, r^* \,\hat{\}}$.

We now define the notion of equality between two null values and between a null value and a non-null value and then justify our definition.

Definition 2.5. When testing for equality of two values, v_1, v_2, be they values of zero order or higher order attributes, we consider the following rule, referred to as the *inequality rule for nulls*:

(1) if $v_1 \cong null$ and $\neg(v_2 \cong null)$ then $v_1 \neq v_2$;

(2) if $v_1, v_2 \cong null$ then $v_1 \neq v_2$.

The above choice of inequality rule for nulls can be justified as follows: when two null values appearing in a nested relation are updated they may be replaced by two distinct non-null values. We note that our model of a single generic unmarked null is less expressive than a model of incomplete information with marked nulls [IMIE84] due to the inequality rule for nulls. This can be shown more formally by mapping each occurrence of *null* in a nested relation onto a unique marked null and then disallowing any marked nulls to be equated. On the other hand, marked nulls are more expensive to maintain and do not always provide more information in the database. Furthermore, we maintain that our formalism of having only a single generic unmarked null is simpler than a formalism that uses marked nulls.

We observe that due to the inequality rule for nulls, two nested relations r_1 and r_2 over a NRS, say R(T), may be identical (and thus information-wise equivalent) yet not equal. We denote this identity by $r_1 \equiv r_2$.

If all the information in a tuple (subtuple) is missing, i.e. all the components of the tuple (subtuple) contain *null*, then we call such a tuple (subtuple) a *null tuple* (*null subtuple*). In the sequel, we shall use the term *null tuple* in a generative sense, i.e. it will mean either a *null tuple* or a *null subtuple*. We further overload our generic null by denoting such a null tuple by *null*.

2.3. The Running Example

In this subsection we give the example that will be used throughout the paper. Let U = {CHILD, CLASS, DAY, DEPT, EXAM, MAJOR, PROJECT, SALARY, STUDENT, TEXT, TUTOR} be the universal set of attributes. The semantics of U are as follows: a STUDENT is enrolled in a department (DEPT) and MAJORs in one or more subjects within the department. A STUDENT attends several CLASSes, each CLASS having several EXAMs during the academic year, which the STUDENT has to sit, and one or more PROJECTs, which the STUDENT has to complete. Each CLASS has several TUTORs and several TEXT books. Also, each TUTOR gives the lectures for the CLASSes the TUTOR teaches on one or more prearranged DAYs. Independently a TUTOR has a SALARY and may have zero or more CHILDren.

Now, let F = $\{T_1, T_2, T_3\}$ be the scheme forest for the running example. The scheme tree T_1 is shown in Figure 2.1, the scheme tree T_2 is shown in Figure 2.5, and the scheme tree T_3 is the scheme tree T shown in Figure 1.1. The NDS, over U, is **R**(F) = {R(T_1), R(T_2), R(T_3)}.

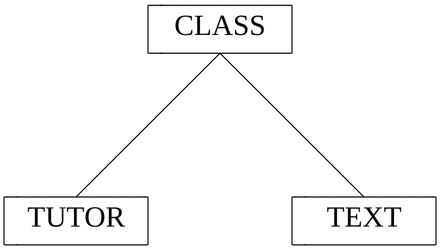

Fig. 2.5. The scheme tree T_2.

Let d* = $\{r_1^*, r_2^*, r_3^*\}$ be a nested database, over the NDS, **R**(F). The nested relation, r_1^*, is shown in Figure 2.2, the nested relation, r_2^*, is shown in Figure 2.6, and the nested relation, r_3^*, is shown in Figure 1.2.

CLASS	(TUTOR)*	(TEXT)*
	TUTOR	TEXT
databases	Robert	Date Ullman
programming	Hanna Richard	Knuth
first-order	*null*	Mendelson
french	Martine	*null*
hebrew	*null*	Bible
null	*null*	Lenin Dostoyevsky

Fig. 2.6. The nested relation r_2^* over R(T_2).

3. THE NULL EXTENDED NESTED RELATIONAL ALGEBRA

In this section we introduce the operators of the null extended algebra [LEVE91] necessary for our formalism in Sections 4 and 5 of null extended data dependencies and the extended chase procedure.

In Subsection 3.1 we define the null extended nest and null extended unnest operators. In Subsection 3.2 we define the null extended union operator. In Subsection 3.3 we define the null extended projection operator and the concept of X-total tuples in nested relations. Finally, in

Subsection 3.4 we define *joinability* in nested relations and the null extended join operator.

In order to ascertain if our null extended algebra is reasonable, we use the notions of *faithfulness* and *preciseness* (cf. [MAIE83, ROTH89]). A null extended operator is *faithful* if it gives the same result, when operating on flat relations, as the corresponding standard operator defined in [CODD79, MAIE83, ULLM88] and taking into account the inequality rule for nulls. A null extended operator is *precise* if unnesting the result, after having applied the null extended operator, gives the same result as first unnesting and then applying the corresponding standard operator to the resulting flat relation(s).

Let *op* be a standard operator and op^{ne} be the corresponding null extended operator. The formal definitions of faithfulness and preciseness now follow.

We say that op^{ne} is *faithful* to *op* if one of the following two conditions holds:

(1) when *op* and op^{ne} are unary operators, $op(r) \cong op^{ne}(r)$, for every flat relation, r, for which $op(r)$ is defined;

(2) when *op* and op^{ne} are binary operators, $r \; op \; s \cong r \; op^{ne} \; s$, for all flat relations, r and s, for which $r \; op \; s$ is defined.

We say that op^{ne} is a *precise* generalisation of *op* relative to unnesting (or simply op^{ne} is a *precise* generalisation of *op*) if one of the following two conditions holds (the operator UNNEST*, denoted by μ^*, is given after Definition 3.2):

(1) when *op* and op^{ne} are unary operators, $\mu^*(op^{ne}(r^*)) \cong op(\mu^*(r^*))$, for every nested relation, r*, for which $op^{ne}(r^*)$ is defined;

(2) when *op* and op^{ne} are binary operators, $\mu^*(r^* \; op^{ne} \; s^*) \cong \mu^*(r^*) \; op \; \mu^*(s^*)$, for all nested relations, r* and s*, for which $r^* \; op^{ne} \; s^*$ is defined.

3.1. Null Extended Nest and Null Extended Unnest

In this subsection we define the two restructuring operators, NEST and UNNEST, in the presence of *nulls*. Our definitions take into account the inequality rule for nulls in Definition 2.5 by using information-wise equivalence instead of equality in our definitions as opposed to Roth et al. [ROTH89] wherein the definitions of NEST and UNNEST remain unaltered when *nulls* are present in nested relations.

We now give the formal definitions of the null extended NEST, denoted as ν, and of the null extended UNNEST, denoted as μ.

Definition 3.1. Let r* be a nested relation over R(T) and let $(Y \neq \Lambda) \subseteq R(T)$. $\nu_Y(r^*)$ is a nested relation over (R(T) - Y)(Y)* such that a tuple $w \in \nu_Y(r^*)$ if and only if

(1) $\exists\, t \in r^*$ such that $t[R(T) - Y] \cong w[R(T) - Y]$;

(2) $w[(Y)^*] \equiv \{t'[Y] \mid t' \in r^* \text{ and } t'[R(T) - Y] \cong w[R(T) - Y]\}$.

We have excluded the case when $Y = \Lambda$, i.e. $\nu_\Lambda(r^*)$, since as we have noted after Definition 2.2 we do not retain empty subtrees in scheme trees in this case.

Definition 3.2. Let r* be a nested relation over R(T) and let $(Y)^* \in H(R(T))$. $\mu_{(Y)^*}(r^*)$ is a nested relation over (R(T) - (Y)*)Y such that a tuple $t \in \mu_{(Y)^*}(r^*)$ if and only if $\exists\, w \in r^*$ such that $t[R(T) - (Y)^*] \cong w[R(T) - (Y)^*]$ and $t[Y] \in w[(Y)^*]$.

The null extended UNNEST* operator [LEVE91] (cf. [THOM86]), denoted by μ^*, transforms any nested relation, r*, into a flat relation. Thomas and Fischer [THOM86] showed that the order of unnesting does not affect the resulting flat relation, $\mu^*(r^*)$.

From now on, for the sake of not overburdening the reader, we shall simply call the null extended NEST, NEST, the null extended UNNEST, UNNEST and the null extended UNNEST*, UNNEST*.

Example 3.1. Let r be the flat relation over ABC, shown in Figure 3.1. Then, $r^* \cong \nu_C(r)$ is the nested relation, r*, over AB(C)*, shown in Figure 3.2, and $\mu_{(C)^*}(r^*)$ is the flat relation, shown in Figure 3.1.

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
null	b_2	c_2
null	b_2	c_3
a_2	*null*	c_1
null	*null*	c_4
null	*null*	c_5

Fig. 3.1. The flat relation, r $\cong$ $\mu_{(C)*}$(r*).

A	B	(C)*
		C
a_1	b_1	c_1 c_2
null	b_2	c_2 c_3
a_2	*null*	c_1
null	*null*	c_4 c_5

Fig. 3.2. The nested relation, r* $\cong$ ν_C(r).

For the nested relation r_1^* of the running example, shown in Figure 2.2, we have the flat relation, r $\cong$ μ*(r_1^*), which is shown in Figure 3.3. We note that r is reduced, since the tuple <*null*, philosophy, *null*, *null*, mid, *null*>, which is less informative than the sixth tuple in r, has been removed from μ*(r_1^*).

We now briefly justify our approach using information-wise equivalence, in contrast to the approach of [ROTH89], wherein equality is used in the definitions of NEST and UNNEST. Using the definition of [ROTH89] ν_C(r) would result in a nested relation, say r′*, comprising the first tuple of r*, shown in Figure 3.2, and all the tuples of the relation r, shown in Figure 3.1, apart from the first and second tuples of r, with the difference that the C-values of these tuples become singletons.

Two advantages follow from our approach. Firstly, the result of our NEST operator is a less redundant representation as can be seen by comparing r*, shown in Figure 3.2, with r′* described above. In addition to this compactness the DBMS can internally consider *nulls* as normal values

STUDENT	DEPT	MAJOR	CLASS	EXAM	PROJECT
Iris	CS	computing	databases	mid	1NF
Iris	CS	computing	databases	final	1NF
Iris	CS	computing	programming	final	*null*
Mark	CS	maths	databases	final	NF2
Mark	CS	maths	databases	final	UR
David	philosophy	logic	first-order	mid	prolog
null	philosophy	*null*	*null*	final	*null*
null	philosophy	*null*	first-order	*null*	functions
null	philosophy	*null*	first-order	*null*	predicates
Naomi	*null*	languages	french	mid	*null*
Naomi	*null*	languages	*null*	final	Moscow
Naomi	*null*	languages	hebrew	*null*	Genesis

Fig. 3.3. The flat relation r $\cong \mu^*(r_1^*)$.

when nesting and unnesting takes place, without the need to make special provision for the null extended NEST and null extended UNNEST operators. Secondly, we claim that r* is a better representation of ν_C(r), since $r'^* \sqsubseteq r^*$ but $\neg(r^* \sqsubseteq r'^*)$. Intuitively, this fact means that r* contains the maximal information emanating from the relation, r, of Figure 3.1.

Roth et al. [ROTH89] raise the question of how to update a nested relation such as r* of Figure 3.2, since it is not known whether *null* is to be replaced by one value, or by more than one value (each such value corresponding to a member of the higher order attribute value). Our justification for the more compact and flexible representation of null extended nested relations is that for updates to be correct they must be specified precisely. For example, if in the second tuple, say t_2, of r*, the null value for A is to be replaced by the non-null value, say a_1, then the (C)*-values that are affected by this update must be specified. Thus, the effect of updating t_2, where (C)* = $\{<c_2>\}$, is to replace t_2 by the two tuples, $\{<a_1, b_2, \{<c_2>\}, <null, b_2, \{<c_3>\}\}$. We claim that a precise update specification must be given whether the updated value is null or non-null. This is due to the fact that nested relations contain more semantic information than their counterpart flat relations. Further discussion and justification of our approach can be found in Levene and Loizou [LEVE89a, LEVE90a].

Incorporating the empty set into nested relations has been a controversial issue, as it is not clear what the result of unnesting the empty set should be. In our model we interpret the empty set, $\emptyset$, over Λ, as an "undefined nested relation". Otherwise, for a non-empty NRS, say R(T), we do not provide for the empty set to be a tuple or subtuple of a nested relation. In cases, when the semantics of *null* coincide with *value does not exist* we interpret $\emptyset$ as $\{null\}$ thus avoiding loss of information when unnesting the empty set as is the case in [ABIT86]. This approach corresponds to the interpretation of the empty set as *value does not exist* [ABIT86, LERA86, MAKI77].

3.2. Union in Nested Relations

We begin by defining the *null extended union* operator, $\cup^{ne}$, which returns the union of two nested relations when tuples are considered as indivisible units.

Definition 3.3. The *null extended union*, $\cup^{ne}$, of two nested relations, r_1 and r_2, over R(T), is defined by:

$$r_1 \cup^{ne} r_2 \equiv \{ \ t \mid t \in r_1 \textbf{ or } t \in r_2 \ \}.$$

(Cf. Zaniolo [ZANI84], Roth et al. [ROTH89], Thomas and Fischer [THOM86], Gyssens et al. [GYSS89] and Van Gucht and Fischer [VANG88].)

A	(B)*	(C)*	(D	(E)*)*
	B	C	D	(E)*
				E
a_1	b_1 b_2	c_1	d_1	*null*
a_1	b_3	*null*	d_2	e_1
a_2	*null*	c_2	*null*	e_1
			d_2	e_2
null	b_2	c_2	*null*	e_2

Fig. 3.4. The nested relation r_1.

Example 3.2. Let R(T) = A(B)*(C)*(D(E)*)* be a NRS and let r_1, r_2 be two nested relations over R(T) shown, respectively, in Figures 3.4 and 3.5. The null extended union, r* $\cong r_1 \cup^{ne} r_2$, is shown in Figure 3.6.

A	(B)*	(C)*	(D	(E)*)*
	B	C	D	(E)*
				E
a_1	b_2	c_1	d_1	*null*
a_1	b_3	*null*	d_2	e_1
null	b_2	c_2	d_2	e_1
			d_2	e_2

Fig. 3.5. The nested relation r_2.

A	(B)*	(C)*	(D	(E)*)*
	B	C	D	(E)*
				E
a_1	b_1 b_2	c_1	d_1	*null*
a_1	b_3	*null*	d_2	e_1
a_2	*null*	c_2	*null*	e_1
			d_2	e_2
null	b_2	c_2	d_2	e_1
			d_2	e_2

Fig. 3.6. The nested relation r* $\cong r_1 \cup^{ne} r_2$.

THEOREM 3.1 [LEVE91]. *The null extended union operator is faithful and precise*. □

Example 3.3. It can easily be verified that for the nested relations, r_1, r_2, shown in Figures 3.4 and 3.5, respectively, the result of Theorem 3.1 holds; thus $\mu^*(r_1 \cup^{ne} r_2) \cong \mu^*(r_1) \cup \mu^*(r_2)$, where $r_1 \cup^{ne} r_2$ is shown in Figure 3.6.

3.3. Projection in Nested Relations

In this subsection we define the projection of a NRS, R(T), onto a subset of its associated set of attributes, S(T); we call the result a *projected NRS*. We then generalise the projection operator to the *null extended projection* operator, denoted as Π^{ne}, defined over nested relations. Lastly, we define the concept of X-total tuples, i.e. tuples which do not contain any null values over a set of attributes, $X \subseteq U$.

We begin by defining a projected NRS.

Definition 3.4. A *projected NRS*, R(T′), of R(T) is defined recursively by:

(1) if T′ = ∅, then R(T′) is a projected NRS of R(T);

(2) if R(T) = X(R(T_1))*(R(T_2))* ... (R(T_s))*, where X = ATT(ROOT(T)), T_1, T_2, ..., T_s, s ≥ 1, denote the first level subtrees of the scheme tree T and R(T'_1), R(T'_2), ..., R(T'_s) are projected NRSs of R(T_1), R(T_2), ..., R(T_s), respectively, then Y(R(T'_1))*(R(T'_2))* ... (R(T'_s))* is a projected NRS of R(T), with Y ⊆ X.

(Cf. Bidoit [BIDO87].)

We observe that in the above definition we include the case where Y = Λ; we further remark that the empty string, Λ, as before is retained in the representation of R(T′) only when it is the root of a projected NRS which has at least one non-empty subtree. For this reason *projected NRSs* are more general than the *projected formats* of Bidoit [BIDO87], since no restrictions are placed on the set of attributes of S(T) over which the projection takes place.

Next we define a useful syntax for projected NRSs.

Definition 3.5. Given a NRS, R(T), and a set of attributes X ⊆ S(T), the projection of R(T) onto X, denoted by R(T)[X], is defined by:

R(T)[X] = R(T′), where R(T′) is a projected NRS of R(T) **and** S(T′) = X.

It can easily be seen that R(T)[X] is unique, and also that if Y $\subseteq$ X, then (R(T)[X])[Y] is unique, i.e. R(T)[Y] = (R(T)[X])[Y]. The next example illustrates Definitions 3.4 and 3.5.

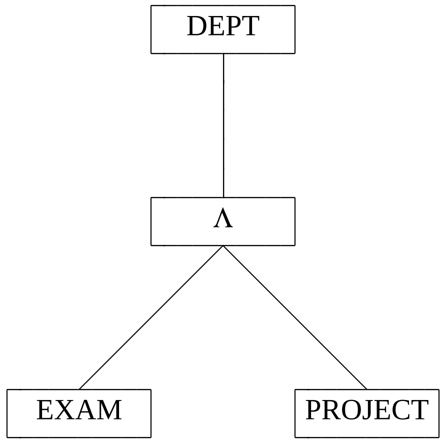

Fig. 3.7. T_1 projected onto {DEPT, EXAM, PROJECT}.

Example 3.4. Let R(T'_1) be the projection of R(T_1), where T_1 shown in Figure 2.1, onto {DEPT, EXAM, PROJECT}, i.e. R(T'_1) = R(T_1)[{DEPT, EXAM, PROJECT}]. Then, R(T'_1) = DEPT(Λ (EXAM)* (PROJECT)*)*, where T'_1 is shown in Figure 3.7.

We are now ready to define the *null extended projection* operator, Π^{ne}. Given a nested relation r* over a NRS, R(T), $\Pi^{ne}{}_Y$(r*), where Y $\subseteq$ S(T), informally returns the largest subset of r*, over a projected NRS, R(T′), of R(T) with S(T′) = Y such that it contains only tuples of the form t[Y], where t $\in$ r*.

Definition 3.6. Let R(T′) be a projected NRS of R(T) over Y $\subseteq$ S(T), and let r* be a nested relation over R(T). Then the *null extended projection* of r* onto Y, $\Pi^{ne}{}_Y$(r*), is a nested relation, over R(T′), defined recursively by:

(1) `if T consists of a single node and ATT(ROOT(T)) = X, then`
$\Pi^{ne}{}_Y$`(r*)` $\equiv$ `{ t[Y] | t` $\in$ `r* }, where Y` $\subseteq$ `X;`

(2) if $R(T) = X(R(T_1))^*(R(T_2))^* \ldots (R(T_s))^*$, where $X = ATT(ROOT(T))$ and $T_1, T_2, \ldots, T_s$, $s \geq 1$, denote the first level subtrees of T, let $R(T') = X'(R(T'_1))^*(R(T'_2))^* \ldots (R(T'_w))^*$, where $X' = ATT(ROOT(T'))$ and $w \leq s$, be a projected NRS of R(T). Then,

$\Pi^{ne}{}_Y(r^*) \equiv \{t \mid \exists\ t' \in r^*$ **and** $t[X'] \cong t'[X']$ **and**

$\forall\ j \in \{1,2,\ldots,w\}$, $w \geq 1$, if $R(T'_j)$ is the projected NRS of $R(T_i)$, $i \in \{1,2,\ldots,s\}$, then

$t[(R(T'_j))^*] \equiv \Pi^{ne}{}_{S(T'_j)}(t'[(R(T_i))^*])\}$.

(Cf. Abiteboul and Bidoit [ABIT86] and Bidoit [BIDO87].)

We note that (1) of Definition 3.6 corresponds to the definition of the standard projection operator for flat relations found in Zaniolo [ZANI84].

Let r* be a nested relation over a NRS, R(T), and let t be a tuple of r*. We extend the definition of the Y-value of t, t[Y], where $Y \subseteq S(T)$, by

$\{t[Y]\} \equiv \Pi^{ne}{}_Y(\{t\})$.

Thus, t[Y] is a tuple over a projected NRS, say R(T′), of R(T), with $Y \subseteq S(T)$. However, we note that if $Y \subseteq R(T)$, then t[Y] is the restriction of t to Y.

THEOREM 3.2 [LEVE91]. *The null extended projection operator is faithful and precise*. □

Example 3.5. Let Y = {CLASS, EXAM, PROJECT}. For r_1^* of Figure 2.2, we obtain the nested relation, $r'^* \cong \Pi^{ne}{}_{(CLASS\ EXAM\ PROJECT)}(r_1^*)$, over the projected NRS, R(T′) = $R(T_1)$[{CLASS, EXAM, PROJECT}], as shown in Figure 3.8.

Λ	(CLASS	(EXAM)*	(PROJECT)*)*
	CLASS	(EXAM)*	(PROJECT)*
		EXAM	PROJECT
	databases	mid final	1NF
	programming	final	*null*
	databases	final	NF2 UR
	first-order	mid	prolog
	null	mid final	*null*
	first-order	*null*	functions predicates
	french	mid	*null*
	null	final	Moscow
	hebrew	*null*	Genesis

Fig. 3.8. The nested relation $r'^{*} \cong \Pi^{ne}_{(CLASS\ EXAM\ PROJECT)}(r_1^{*})$.

The next definition formally defines the concept of X-total tuples in a nested relation, for a set of attributes, $X \subseteq S(T)$. Let r* be a nested relation over R(T) and $t \in r^*$. Informally, the *definite portion* of t, DEF(t), returns a set of attributes in S(T) for which the tuple t has no *nulls* in its corresponding attributes in R(T). Let $X \subseteq S(T)$, then we say that the tuple t is X-total if and only if $X \subseteq DEF(t)$.

Definition 3.7. Let r* be a nested relation over a NRS, R(T). Then, for $t \in r^*$, DEF(t) is defined recursively by:

(1) if the scheme tree T conists of a single node, n, and ATT(n) = X, then

DEF(t) = $\{A \mid A \in X$ **and** $\neg(t[A] \cong$ *null*$)\}$;

(2) if X = ATT(ROOT(T)) and $T_1, T_2, \ldots, T_s$, $s \geq 1$, denote the first level subtrees of the scheme tree, T, then

DEF(t) = $DEF(t[X]) \cup (\cup_{i=1}^{s} \{\cap \{DEF(t') \mid t' \in t[(R(T_i))^*]\}\})$.

We note that (1) of the above definition corresponds to the standard definition of X-total tuples for flat relations found in Sagiv [SAGI83], Maier [MAIE83], Maier et al. [MAIE84] and Stein and Maier [STEI85].

Example 3.6. Consider the nested relation $r_1^* = \{t_1, t_2, t_3, t_4, t_5\}$, over R($T_1$), of the running example. For the first tuple, t_1, we have DEF(t_1) = {STUDENT, DEPT, MAJOR, CLASS, EXAM}, since t_1[(PROJECT)*] contains *null*. Thus t_1 is {STUDENT, DEPT, MAJOR, CLASS, EXAM}-total. For the second and third tuples, we have DEF(t_2) = DEF(t_3) = S(T_1), since there are no *nulls* in either t_2 or t_3. Thus t_2 and t_3 are both S(T_1)-total. For the tuple, t_4, we have DEF(t_4) = {DEPT}, since DEPT is the only attribute over which t_4 has no *nulls*. Thus t_4 is DEPT-total. Finally, for the tuple, t_5, we have DEF(t_5) = {STUDENT, MAJOR}, since STUDENT and MAJOR are the only attributes over which t_5 has no *nulls*. Thus t_5 is {STUDENT, MAJOR}-total.

3.4. Join in Nested Relations

In this subsection we first define the concept of a *joinable NDS*, **R**(F), over the universal set of attributes, U = S(F). Intuitively, **R**(F) is joinable if all the NRSs R(T_i) ∈ **R**(F), i=1,2,...,q, can be combined into a single NRS over U, denoted as U(T) and called the *joined NRS* of **R**(F), without violating the definition of a scheme tree. We then generalise the (natural) join operator [CODD79, MAIE83, ULLM88] to the *null extended join* operator, denoted as $\bowtie^{ne}$, which can only be applied to joinable NDSs.

We note that our null extended join is more general than other extended joins defined for nested relations [ABIT86, COLB89, JAES82, THOM86, ROTH89], because in our null extended join attributes are joined at all heights of nodes in the scheme trees of a joinable NDS.

We now formally define a joinable NDS.

Definition 3.8. Let R(T_1) and R(T_2) be NRSs, then R(T_1) and R(T_2) are *joinable NRSs* if and only if there exists a NRS, R(T), over S(T_1) ∪ S(T_2), such that R(T)[S(T_1)] = R(T_1) and R(T)[S(T_2)] = R(T_2). **R**(F) is said to be a *joinable NDS* if and only if for each pair i,j ∈ {1,2,...,q} R(T_i) and R(T_j) are *joinable NRSs*.

We observe that *joinable NDSs* are a generalisation of *compatible formats* [ABIT86], since we do not restrict the NRSs in the joinable NDS to have the same attributes in their root nodes.

The next definition builds on the preceding one in order to characterise the resulting NRS of a joinable NDS.

Definition 3.9. Let **R**(F) be a joinable NDS. Then U(T) is the *joined NRS* of **R**(F) if

(1) S(T) = S(F);

(2) U(T)[S(T_i)] = R(T_i), $1 \le i \le q$.

The following theorem shows that if a NDS, **R**(F), is joinable then this fact implies the existence of a joined NRS, U(T), of **R**(F). Additionally, the theorem also shows that if U(T) is the joined NRS of a NDS, **R**(F), then **R**(F) must be joinable.

THEOREM 3.3 [LEVE91]. *A NDS,* **R***(F), is joinable if and only if there exists a joined NRS, U(T), of* **R***(F).* □

The above, besides many other features, are discussed in [LEVE90a], wherein restructuring of an arbitrary NDS into a joinable NDS is dealt with in detail.

Example 3.7. Let T_2 denote the scheme tree shown in Figure 2.5 and let T'_3 be the scheme tree shown in Figure 3.9. Let F′ = {T_2, T'_3}; then **R**(F′) = {R(T_2), R(T'_3)} is a joinable NDS and U(T), where T is shown in Figure 3.10, is the joined NRS of **R**(F′). On the other hand, let **R**(F″) = {R(T_2), R(T_3)}, where T_3 is shown in Figure 1.1. It can easily be verified that **R**(F″) is **not** a joinable NDS, i.e. there does not exist a joined NRS, say U(T′), of **R**(F″).

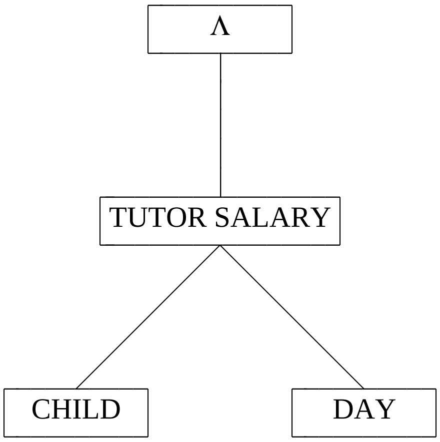

Fig. 3.9. The scheme tree T'_3.

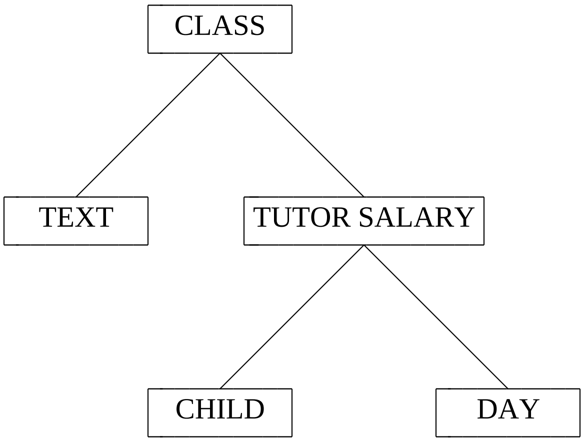

Fig. 3.10. The scheme tree T for U(T).

We are now in a position to define the null extended join operator. Informally, the *null extended join* of two nested relations, r_1 and r_2, over a pair of joinable NRSs, R(T_1) and R(T_2), performs a join operation between the tuples of r_1 and r_2 in such a way that the join takes place recursively, at each height of nodes in the corresponding scheme trees, T_1 and T_2, whilst taking into account common attributes labelling the nodes of the two scheme trees.

Definition 3.10. Let R(T_1), R(T_2), with corresponding nested relations, r_1 and r_2, be joinable NRSs such that R(T)[S(T_1)] = R(T_1) and R(T)[S(T_2)] = R(T_2). The *null extended join* of r_1 and r_2, $r_1 \bowtie^{ne} r_2$, yielding r* over R(T), is defined recursively by:

(1) if the T_i, i = 1,2, consist of single nodes and ATT(ROOT(T_i)) = X_i, then

$$r^* \equiv \{t \mid \exists t_1 \in r_1, \exists t_2 \in r_2 : t_1[X_1 \cap X_2] = t_2[X_1 \cap X_2] \textbf{ and } t[X_1] \cong t_1[X_1] \textbf{ and } t[X_2] \cong t_2[X_2]\};$$

(2) if R(T_i) = X_i(R(T_1^i))*(R(T_2^i))* ... (R($T_{s_i}^i$))*, i = 1,2, where X_i = ATT(ROOT(T_i)) and $T_1^i, T_2^i, \ldots, T_{s_i}^i$, $s_i \geq 1$, denote the first level subtrees of T_i, then

$r^* \equiv \{t \mid \exists t_1 \in r_1, \exists t_2 \in r_2$:

(2.1) ($t[X_1X_2] \equiv t_1[X_1] \bowtie^{ne} t_2[X_2]$ **and**

$t[X_1X_2] \neq \varnothing$ if ($X_1 \neq \Lambda$ **or** $X_2 \neq \Lambda$)); **and**

(2.2) (let j $\in$ {1,2,...,s_1}, k $\in$ {1,2,...,s_2}, then

(2.2.1) $\forall$ j, k such that S(T_j^1) $\cap$ S(T_k^2) $\neq \varnothing$:

(t[(R(T)[S(T_j^1) $\cup$ S(T_k^2)])*] $\equiv$

t_1[(R(T_j^1))*] $\bowtie^{ne}$ t_2[(R(T_k^2))*]) $\neq \varnothing$; **or**

(2.2.2) $\forall$ j such that $\forall$ k S(T_j^1) $\cap$ S(T_k^2) = $\varnothing$:

t[(R(T_j^1))*] $\equiv t_1$[(R(T_j^1))*]; **or**

(2.2.3) $\forall$ k such that $\forall$ j S(T_j^1) $\cap$ S(T_k^2) = $\varnothing$:

t[(R(T_k^2))*] $\equiv t_2$[(R(T_k^2))*])}.

(Cf. join in Abiteboul and Bidoit [ABIT86] and Colby [COLB89], extended natural join in Roth et al. [ROTH89], intersection join in Jaeschke and Schek [JAES82] and Thomas and Fischer [THOM86], and unnest-join in Korth [KORT88].)

We note that (1) of Definition 3.10 corresponds to the standard definition of the join operator found in [CODD79, MAIE83, ULLM88].

Example 3.8. Let R(T_1) = A(B)*(C)*(D(E)*)* and R(T_2) = Λ(BF)*(G)*(Λ(E)*)* be joinable NRSs such that R(T)[S(T_1)] = R(T_1) and R(T)[S(T_2)] = R(T_2), where R(T) = A(BF)*(C)*(G)*(D(E)*)*. Also, let r_1 be the nested relation, over R(T_1), shown in Figure 3.11, and let r_2 be the nested relation, over R(T_2), shown in Figure 3.12 The result of the null extended join, r* ≅ $r_1 \bowtie^{ne} r_2$, is shown in Figure 3.13.

A	(B)*	(C)*	(D	(E)*)*
	B	C	D	(E)*
				E
a_1	b_1 b_2 b_3	c_1 c_2	d_1	e_1 e_2
a_2	b_1	*null*	d_2	e_1
a_3	b_1 b_2	c_1	*null*	*null*
null	b_2	c_2 c_3	*null* d_3	e_2 *null*

Fig. 3.11. The nested relation r_1.

Λ	(BF)*	(G)*	(Λ	(E)*)*
	BF	G	Λ	(E)*
				E
	$b_1\ f_1$ $b_2\ f_1$	g_1 g_2		e_1 e_2

Fig. 3.12. The nested relation r_2.

A	(BF)*	(C)*	(G)*	(D	(E)*)*
	BF	C	G	D	(E)*
					E
a_1	$b_1\ f_1$ $b_2\ f_1$	c_1 c_2	g_1 g_2	d_1	e_1 e_2
a_2	$b_1\ f_1$	*null*	g_1 g_2	d_2	e_1
null	$b_2\ f_1$	c_2 c_3	g_1 g_2	*null*	e_2

Fig. 3.13. The null extended join, r* ≅ $r_1 \bowtie^{ne} r_2$.

THEOREM 3.4 [LEVE91]. *The null extended join operator is faithful and precise.* □

Example 3.9. It can easily be verified that the result of Theorem 3.4 holds for r* ≅ $r_1 \bowtie^{ne} r_2$, shown in Figure 3.13; thus μ*(r*) ≅ μ*(r_1) $\bowtie$ μ*(r_2).

By utilising our null extended join we can define the *null extended Cartesian product*, denoted as $\times^{ne}$, and the *null extended intersection*, denoted as $\cap^{ne}$. (Cf. [ABIT86, ROTH89].) The null extended join reduces to the null extended Cartesian product when S(T_1) ∩ S(T_2) = ∅, and it reduces to the null extended intersection when R(T_1) = R(T_2).

Example 3.10. Let r'^*_3 ≅ $\nu_{R(T_3)}(r^*_3)$ be the nested relation of Figure 1.2, over R(T'_3), where T'_3 is shown in Figure 3.9. Now, let d′* = {r^*_2, r'^*_3} be a nested database over the NDS, **R**(F′), of Example 3.7, where r^*_2 is shown in Figure 2.6. Then, the null extended join, $r^*_2 \bowtie^{ne} r'^*_3$, over U(T), where T is shown in Figure 3.10, is given in Figure 3.14.

CLASS	(TEXT)*	(TUTOR	SALARY	(CHILD)*	(DAY)*)*
	TEXT	TUTOR	SALARY	(CHILD)*	(DAY)*
				CHILD	DAY
databases	Date Ullman	Robert	12000	Hanna Brian	Monday Thursday
programming	Knuth	Hanna	14000	Annette Ada	*null*
french	*null*	Martine	*null*	*null*	*null*

Fig. 3.14. The null extended join $r^*_2 \bowtie^{ne} r'^*_3$.

4. NULL EXTENDED DATA DEPENDENCIES AND THEIR ASSOCIATED RULES

Herein we define semantics for the null extended nested relational model in terms of the new class of *null extended data dependencies*, which are integrity constraints that hold in nested relations. We also consider the counterpart of the class of null extended data dependencies, namely, the class of *null data dependencies*, which hold in the flat relations emanating from the said nested relations.

In Subsection 4.1 we generalise FDs so that they hold in flat relations containing *nulls*; we call these generalised FDs *null functional dependencies* (NFDs) [LIEN82, ATZE86]. Then we redefine NFDs in the context of nested relations.

In Subsection 4.2 we extend the notion of NFDs from flat relations to nested relations in a natural way, namely, instead of zero order attributes appearing on the right-hand side of a NFD we now have a higher order attribute, for example, $X \rightarrow (Y)^*$. Thus, instead of requiring information-wise equivalence over the zero order attribute on the right-hand side of a NFD, we now require information-wise equivalence over $(Y)^*$. These extended NFDs, in the context of nested relations, are called *null extended functional dependencies* (NEFDs) (cf. [FISC85, JAES82, MAKI77, MIUR86, THOM86, VANG88]).

Join dependencies (JDs), as integrity constraints, are of major importance, since they capture the notion of *lossless decomposition* [BEER81, MAIE83, SCIO82, ULLM88], the *multivalued dependency* (MVD) being a special case of the JD when the cardinality of the decomposition is just two [MAIE83, ULLM88]. In Subsection 4.3 we generalise JDs that hold in flat relations to *null join dependencies* (NJDs) (cf. [JAJO90]). As a special case of NJDs we have the *null multivalued dependency* (NMVD) [LIEN79, LIEN82] when the cardinality of the decomposition is two. In addition, we extend NJDs, which hold in flat relations, to nested relations; we call such extended JDs *null extended join dependencies* (NEJDs). The extension of NJDs to NEJDs is achieved by utilising the null extended join operator given in Subsection 3.4. As a consequence of this approach we are now able to capture the novel notion of losslessness in nested relations, which we call *null extended lossless decomposition*.

For each of the null extended data dependencies, the NFD, the NEFD and the NEJD, we define associated rules, which are used in Section 5, wherein the extended chase is defined in order to enforce and test the satisfaction of a set of null extended data dependencies in a nested relation.

In Subsection 4.4 we define null hierarchical dependencies, which comprise a set of NEFDs that are naturally represented in a scheme tree. We show that this set of NEFDs for a scheme tree, T, denoted as FD(T), can equivalently be explained in terms of a set of NMVDs, denoted by MVD(T). Furthermore, this set of NMVDs is equivalent to the single JD, $\bowtie$[P(T)], represented by the path set of T.

Next, we define the notion of implication for a set of null extended data dependencies and the counterpart set of null data dependencies. Let D be a set of null data dependencies, and let SAT(D) denote the set of flat relations, over U, that satisfy D. We say that D implies a single null data dependency, d_i, written in the form of D $\models d_i$ if and only if SAT(D) $\subseteq$ SAT(d_i). Correspondingly, let D* be a set of null extended data dependencies, and let SAT(D*) denote the set of nested relations, over U(T), that satisfy D*. We say that D* implies a single null extended data dependency, d_i^*, written in the form of D* $\models d_i^*$ if and only if SAT(D*) $\subseteq$ SAT(d_i^*). Similarly, we define D* $\models d_i$ if and only if {μ*(r*) | r* $\in$ SAT(D*)} $\subseteq$ SAT(d_i) and D $\models d_i^*$ if and only if SAT(D) $\subseteq$ {μ*(r*) | r* $\in$ SAT(d_i^*)}, where D is the counterpart set of null data dependencies corresponding to D* and $d_i \in$ D.

A set D_1 of null extended data dependencies (or null data dependencies) implies a set D_2 of null extended data dependencies (or null data dependencies), denoted by $D_1 \models D_2$ if and only if $D_1 \models$ d for all null extended data dependencies (or null data dependencies), d $\in D_2$. Finally, D_1 is equivalent to D_2, denoted by $D_1 \approx D_2$, if and only if $D_1 \models D_2$ and $D_2 \models D_1$.

4.1. Null Functional Dependencies

In this subsection we generalise the standard FD that holds in flat relations without *nulls* to the NFD that holds in (null extended) flat relations. We then redefine the NFD, a member of the class of null data dependencies, over a set of attributes labelling a node in a scheme tree, T, in order that it hold in a nested relation over a NRS, R(T). Such a NFD is a member of the class of null extended data dependencies. We then discuss some characteristics of NFDs and define a NFD-rule used in Section 5 for computing the extended chase of a nested relation with respect to a set of null extended data dependencies.

Definition 4.1. Let r be a flat relation over a set of attributes W $\subseteq$ U and let X, A $\subseteq$ U, where A is a single attribute. Then the NFD, X $\rightarrow$ A, holds in r if and only if whenever there exist tuples t_1, $t_2 \in$ r such that t_1[X] = t_2[X] (i.e. t_1 and t_2 are X-total), then t_1[A] $\cong$ t_2[A] (i.e. t_1 and t_2 are both A-total or t_1[A] $\cong$ t_2[A] $\cong$ *null*).

(Cf. [ATZE86, LIEN82].)

We now redefine the NFD in the context of nested relations.

Definition 4.2. Let r* be a nested relation over a NRS, R(T), and let X, A $\subseteq$ ATT(n), where n is a node in T and A is a single attribute. Then the NFD, X $\rightarrow$ A, holds in r* if and only if X $\rightarrow$ A holds in μ*($\Pi^{ne}{}_{XA}$(r*)).

If A $\in$ X then the NFD, X $\rightarrow$ A, is said to be a *trivial* NFD, otherwise it is said to be a *non-trivial* NFD.

We denote the set of non-trivial NFDs, which are represented in the nodes of a scheme tree, T, by FF(T), and the set of non-trivial NFDs, which are represented in the nodes of a scheme forest, F = {T_1, T_2, ..., T_q}, by FF(F), where FF(F) = $\cup_{i=1}^{q}$ FF(T_i).

Example 4.1. For the scheme tree, T_1, shown in Figure 2.1, we have ATT(n) = {STUDENT, DEPT}, where n is the root node of T_1. The NFD, STUDENT $\rightarrow$ DEPT, holds in the nested relation, r_1^*, shown in Figure 2.2, over the NRS, R(T_1), and thus FF(T_1) = {STUDENT $\rightarrow$ DEPT}. Similarly, for the scheme tree, T_3, shown in Figure 1.1, we have ATT(n) = {TUTOR, SALARY}, where n is the root node of T_3. The NFD, TUTOR $\rightarrow$ SALARY, holds in the nested relation, r_3^*, shown in Figure 1.2, over the NRS, R(T_3), and thus FF(T_3) = {TUTOR $\rightarrow$ SALARY}. Thus, for the running example, FF(F) = FF(T_1) $\cup$ FF(T_3).

For flat relations without *nulls*, Definition 4.2 reduces to the standard definition of FD [MAIE83, ULLM88]. A sound and complete set of inference rules for NFDs is provided by Lien [LIEN82]; this set does not include transitivity [MAIE83, ULLM88], which may not hold in the presence of nulls [ATZE86, LIEN82]. For example, the well-known flat relation [ATZE86], shown in Figure 4.1, satisfies the NFDs, {A $\rightarrow$ B, B $\rightarrow$ C}, but violates the NFD, A $\rightarrow$ C.

A	B	C
a_1	*null*	c_1
a_1	*null*	c_2

Fig. 4.1. A relation which violates transitivity for NFDs.

We close this subsection with the definition of the NFD-rule for a NFD, X $\rightarrow$ A, with respect to a nested relation, r*, over a NRS, R(T).

Definition 4.3. Let r* be a nested relation, over a NRS, R(T), and let FF(T) be the set of NFDs, which are represented in the nodes of the scheme tree, T. Then the *NFD-rule* for the NFD, X $\rightarrow$ A $\in$ FF(T), denoted as $RULE_{X \rightarrow A}$(r*), is defined as follows:

Let t_1, t_2 be two XA-tuples in $\mu^*(\Pi^{ne}{}_{XA}(r^*))$ which are not necessarily distinct. If t_1[X] = t_2[X], i.e. t_1 and t_2 are X-total, and $\neg(t_1[A] \cong t_2[A])$, then

(1) if $t_1[A] \leq t_2[A]$, then $RULE_{X \rightarrow A}(r^*) \equiv r'^*$, where r′* is r* after the assignment $t_1[A] := t_2[A]$;

(2) if $t_2[A] \leq t_1[A]$, then $RULE_{X \rightarrow A}(r^*) \equiv r'^*$, where r′* is r* after the assignment $t_2[A] := t_1[A]$;

(3) if $\neg(t_1[A] \leq t_2[A])$ and $\neg(t_2[A] \leq t_1[A])$, then the NFD, X $\rightarrow$ A, does not hold in r*, and $RULE_{X \rightarrow A}(r^*) \equiv \{null\}$, i.e. it is undefined.

4.2. Null Extended Functional Dependencies

In this subsection we extend NFDs holding in flat relations so that they obtain in nested relations, and call them, *null extended functional dependencies* (NEFDs). The NEFD is a member of the class of null extended data dependencies.

The following definition of the NEFD extends Definition 4.1, so that the NFD holds in nested relations; this is effected by allowing higher order attributes on the right-hand side of the

NFD.

Definition 4.4. Let r* be a nested relation over a NRS, R(T). Then the NEFD, X $\rightarrow$ (Y)*, holds in r* if and only if the following recursive definition is satisfied:

(1) if X $\subseteq$ Z(R(T)) and (Y)* $\in$ H(R(T)), then, whenever there exist tuples t_1, t_2 $\in$ r* such that t_1[X] = t_2[X], i.e. t_1, t_2 are X-total, t_1[(Y)*] $\stackrel{=}{\sim}$ t_2[(Y)*]; otherwise

(2) let T_1, T_2, ..., T_s, s $\geq$ 1, denote the first level subtrees of the scheme tree, T. Then for all tuples, t* $\in$ r*, $\exists$ i $\in$ {1,2,...,s} such that the NEFD, X $\rightarrow$ (Y)*, holds in the nested relation, {t*[ATT(ROOT(T))]} $\times^{ne}$ t*[(R(T_i))*], over the NRS, ATT(ROOT(T))R(T_i).

Example 4.2. For the scheme tree, T_1, of the running example shown in Figure 2.1 we have the set of NEFDs, F_1 = {STUDENT,DEPT $\rightarrow$ (MAJOR)*, STUDENT,DEPT $\rightarrow$ (CLASS (EXAM)* (PROJECT)*)*, STUDENT,DEPT,CLASS $\rightarrow$ (EXAM)*, STUDENT,DEPT,CLASS $\rightarrow$ (PROJECT)*}.

For the scheme tree, T_2, of the running example shown in Figure 2.5 we have the set of NEFDs, F_2 = {CLASS $\rightarrow$ (TUTOR)*, CLASS $\rightarrow$ (TEXT)*}. For the scheme tree, T_3, of the running example shown in Figure 1.1 we have the set of NEFDs, F_3 = {TUTOR,SALARY $\rightarrow$ (CHILD)*, TUTOR,SALARY $\rightarrow$ (DAY)*}.

It thus follows that for the scheme forest, F, of the running example, we have the set of NEFDs, F = $F_1 \cup F_2 \cup F_3$. It can easily be verified that F holds for the nested database d* = {r_1^*, r_2^*, r_3^*} of the running example, where r_1^* is shown in Figure 2.2, r_2^* is shown in Figure 2.6 and r_3^* is shown in Figure 1.2.

We now illustrate Definition 4.4 in more detail by showing that STUDENT, DEPT, CLASS $\rightarrow$ (EXAM)* holds in the first tuple, say t*, of r_1^* over the NRS R(T_1). Definition 4.4 part(1) is false, since Z(R(T_1)) = {STUDENT, DEPT} and {STUDENT, DEPT, CLASS} $\not\subseteq$ {STUDENT, DEPT}. For Definition 4.4 part (2) we have

t*[ATT(ROOT(T_1))] = t*[{STUDENT, DEPT}] = <Iris, CS> and

t*[CLASS(EXAM)*] ≅ {<databases, {<mid>, <final>}>, <programming, {<final>}>}.

Now, it can easily be verified that STUDENT, DEPT, CLASS $\rightarrow$ (EXAM)* holds in the nested relation, {t*[{STUDENT, DEPT}]} $\times^{ne}$ t*[CLASS(EXAM)*] ≅

{<Iris, CS, databases, {<mid>, <final>}>, <Iris, CS, programming, {<final>}>},

over STUDENT,DEPT,CLASS(EXAM)*, by Definition 4.4 part (1).

We close this subsection with the definition of the NEFD-rule for a NEFD, X $\rightarrow$ (Y)*, with respect to a nested relation, r*, over a NRS, R(T).

Definition 4.5. Let r* be a nested relation, over a NRS, R(T). Then the *NEFD-rule* for the NEFD, X $\rightarrow$ (Y)*, denoted as $RULE_{X \rightarrow (Y)*}$(r*), is defined recursively as follows :

(1) if X = Z(R(T)) and (Y)* $\in$ H(R(T)), and there exist tuples t_1, t_2 $\in$ r* such that t_1[X] = t_2[X], i.e. t_1 and t_2 are X-total, and $\neg$(t_1[(Y)*] ≅ t_2[(Y)*]), then $RULE_{X \rightarrow (Y)*}$(r*) $\equiv$ r′*, where r′* is r* after the assignment t_1[(Y)*], t_2[(Y)*] := t_1[(Y)*] $\cup^{ne}$ t_2[(Y)*]; otherwise

(2) let T_1, T_2, ..., T_s, s $\geq$ 1, denote the first level subtrees of the scheme tree, T, and let t* be a tuple in r*. Then we recursively apply the NEFD-rule for the NEFD, X $\rightarrow$ (Y)*, to the nested relation, r'^*_i $\equiv$ {t*[ATT(ROOT(T))]} $\times^{ne}$ t*[(R(T_i))*], over the NRS, ATT(ROOT(T))R(T_i), where i $\in$ {1,2,...,s}, i.e. we recursively apply $RULE_{X \rightarrow (Y)*}(r'^*_i)$.

4.3. Null Extended Join Dependencies

In this subsection we define the NJD, a member of the class of null data dependencies. We justify our definition and discuss the effect of nulls on the inference rules for JDs given in [BEER81].

We then extend NJDs to NEJDs, by incorporating the null extended join operator, given in Subsection 3.4, into the definition of the NJD. The NEJD, a member of the class of null extended data dependencies, is a new data integrity constraint for nested relations and enables us to capture the novel notion of lossless decomposition in nested relations. We then investigate the properties of NEJDs with reference to NJDs. In particular, we show that the NEJD is faithful to the NJD in the sense that a NEJD reduces to a NJD when only flat relations are considered. Finally, we show that the NEJD is a precise generalisation of the NJD in the sense that the NEJD holds in a nested relation iff its counterpart NJD holds in the corresponding flat relation.

Herein we employ the following useful notation for FDSs found in [BEER81]. *Let* **R** *and* **S** *be two FDSs over U*. We say that **S** *covers* **R**, if for every FRS, $R_i \in$ **R**, there exists a FRS, $S_j \in$ **S**, such that $R_i \subseteq S_j$. The set of all FDSs that cover **R** is denoted by COVER(**R**), and MANY(**R**) denotes the set of attributes that appear in at least two FRSs in **R**.

Let **S** = $\{S_1, S_2, ..., S_k\}$ be a FDS such that **S** $\subseteq$ **R**. Then **S** is a *connected subset* of **R** if and only if there exists a permutation, say $(S_1^j, S_2^j, ..., S_k^j)$, of **S** such that $S_i^j \cap S_{i+1}^j \neq \emptyset$, 1 ≤ i < k. The FDS, **S**, is a *covering subset* of the FDS, **R**, if each FRS, $S_i \in$ **S**, is a set of attributes over a connected subset, say $\mathbf{S}_i$, of **R** and such that $\mathbf{R} = \cup_i \mathbf{S}_i$. We denote the set of all *covering subset FDSs* of **R** by SUBSET(**R**). We note that if **S** ∈ SUBSET(**R**), then **S** ∈ COVER(**R**), but the converse is, in general, false.

We say that n not necessarily distinct tuples, $t_1, t_2, ..., t_n \in$ r, are *combinable* on a covering subset FDS, **S** ∈ SUBSET(**R**), where (n = |**S**|) ≤ (q = |**R**|), with the resulting tuple, t, over U, if $t[S_i] \cong t_i[S_i]$, $S_i \in$ **S**, 1 ≤ i ≤ n, and t[MANY(**S**)] is MANY(**S**)-total. (Cf. [MAIE79, STEI85].)

In the sequel, whenever the parameters are fully understood from context, we abbreviate the above to: *the tuples*, $t_1, t_2, ..., t_n \in$ r, *are combinable with the resulting tuple, t, over U.*

Definition 4.6. The NJD, $\bowtie$[**R**], holds in r if and only if whenever there exist n not necessarily distinct tuples, $t_1, t_2, ..., t_n \in$ r, that are combinable with the resulting tuple, t, over the universe, U, then t $\in$ r also holds.

As with JDs, the NJD, $\bowtie[\mathbf{R}]$, is said to be a *trivial* NJD if q = 1, otherwise it is said to be a *non-trivial* NJD.

It can easily be seen that the *null multivalued dependency* (NMVD) [LIEN79, LIEN82] is a special case of the NJD, i.e. when q = 2. Also, it can easily be verified that for flat relations without *null*, Definition 4.6 reduces to the standard definition of the JD [MAIE83, ULLM88]. Furthermore, it can be verified that the *covering* and *projection* rules for JDs [BEER81] are also sound for NJDs over flat relations with *null* due to Definition 4.6. We note that we can formally show that the said covering rule holds by using a result from Beeri and Vardi [BEER81], wherein it was shown that a FDS, **S**, covers a FDS **R** if and only if **S** can be obtained from **R** by repetitively, either adding a FRS to **R**, or by adding an attribute to a FRS already in **R**.

On the other hand, the *substitution* rule for JDs [BEER81] does not, in general, hold for NJDs, since, as is the case with NMVDs, transitivity of NJDs may not hold in the presence of *null*. A simple example illustrates this: let D = {AB $\to\to$ C | D, A $\to\to$ B | CD} be a set of NMVDs (using the standard notation for MVDs [MAIE83, ULLM88]) and r $\equiv$ {<a_1, *null*, c_1, d_1>, <a_1, *null*, c_2, d_2>}. It can easily be verified that r $\in$ SAT(D) but r does not satisfy A $\to\to$ C, which would be inferred by using the substitution rule.

We now present a special case of the substitution rule, called the *non-split substitution rule*, which is sufficient for our purposes in this paper. Let **R** be a FDS, over U, and let **S** = {S_1, S_2} be a FDS over the attributes of an $R_i \in$ **R** such that either (MANY(**R**) $\cap$ R_i) $\subseteq$ ($S_1 \cap S_2$) or MANY(**R**) $\cap$ $S_1 = \emptyset$. Then, the *non-split substitution rule* for NJDs is given by

$\{\bowtie[\mathbf{R}], \bowtie[\mathbf{S}]\} \models \bowtie[(\mathbf{R} - \{R_i\}) \cup \mathbf{S}]$.

Let r be a flat relation, over U, that satisfies $\bowtie[\mathbf{R}]$; assume that Π_{R_i}(r) satisfies $\bowtie[\mathbf{S}]$. Also, let **Q** $\in$ (SUBSET((**R** - {R_i}) $\cup$ **S**) $-$ SUBSET(**R**)) be a FDS with |**Q**| = n. We claim that if the tuples, $t_1, t_2, ..., t_n \in$ r, are combinable on **Q** with the resulting tuple, t, over U, then t $\in$ r.

We next outline a proof of the above claim, thereby showing that the non-split substitution rule holds for NJDs over flat relations with *null*.

Assume that $\neg(t \in r)$. Now, since $\bowtie[\mathbf{S}]$ holds in $\Pi_{R_i}(r)$, it follows that there exists a tuple, $t' \in r$, such that $t'[R_i] \cong t[R_i]$ and $\neg(t'[U - R_i] \cong t[U - R_i])$. In addition, we have that $t'[\text{MANY}(\mathbf{R}) \cap R_i] = t[\text{MANY}(\mathbf{R}) \cap R_i]$, since either $(\text{MANY}(\mathbf{R}) \cap R_i) \subseteq (S_1 \cap S_2)$ or $\text{MANY}(\mathbf{R}) \cap S_1 = \emptyset$.

There must now exist a subset of $\{t_1, t_2, ..., t_n\}$ whose tuples are combinable on a covering subset $\mathbf{Q'} \in \text{SUBSET}(\mathbf{R})$ with the resulting tuple $t'_i \in r$ such that $t'_i[U - R_i] \cong t[U - R_i]$ and $t'_i[\text{MANY}(\mathbf{R}) \cap R_i] = t[\text{MANY}(\mathbf{R}) \cap R_i]$. It, therefore, follows that t' and t'_i are combinable on $\{((U - R_i) \cup (\text{MANY}(\mathbf{R}) \cap R_i)), R_i\}$ with the resulting tuple, $t \in r$, which leads to a contradiction.

We now give a brief example of the non-split substitution rule. Assume that D consists of the two NMVDs: STUDENT,DEPT $\rightarrow\rightarrow$ MAJOR | CLASS,EXAM,PROJECT and STUDENT,DEPT,CLASS $\rightarrow\rightarrow$ EXAM | PROJECT. It can easily be verified that by applying the non-split substitution rule, $D \models \bowtie[P(T_1)]$; we remind the reader that $P(T_1)$ is given in Example 2.1.

Next we extend the notions of a covering subset FDS and combinable tuples to a covering subset NDS and combinable tuples of nested relations, respectively. We recall from Subsection 2.1 that if F is a scheme forest, over U, then $\text{FDS}(F) = \{S(T_1), S(T_2), ..., S(T_q)\}$.

Let $\mathbf{R}$(F) be a joinable NDS, over the joined NRS, U(T), and let r* be a nested relation over U(T). Then, the NDS $\mathbf{R}(F')$, where $F' = \{T'_1, T'_2, ..., T'_n\}$ is a scheme forest over the joined NRS, U(T), is a *covering subset* of the NDS, $\mathbf{R}$(F), if $\text{FDS}(F') \in \text{SUBSET}(\text{FDS}(F))$. We denote the set of all covering subset NDSs of $\mathbf{R}$(F) by SUBSET($\mathbf{R}$(F)).

As before, we say that n not necessarily distinct tuples, $t_1, t_2, ..., t_n \in$ r*, are *combinable* on a covering subset NDS, **R**(F′) ∈ SUBSET(**R**(F)), where (n = |**R**(F′)|) ≤ (q = |**R**(F)|), with the resulting tuple, t*, over U(T), if

$$t^* \cong \bowtie^{ne}\ {}_{i=1}^{n}\ (\Pi^{ne}{}_{S(T'_i)}\ (t_i)) \neq \emptyset, \qquad (1)$$

where $S(T'_i) \in$ FDS(F′), $1 \le i \le n$. Consequently, t*[MANY(FDS(F′))] is MANY(FDS(F′))-total, due to Definition 3.10 of $\bowtie^{ne}$.

In the sequel, whenever the parameters are fully understood from context, we abbreviate the above to: *the tuples, $t_1, t_2, ..., t_n \in r^*$, are combinable with the resulting tuple, t*, over U(T).*

We are now in a position to extend the NJD, defined over flat relations, to the NEJD, defined over nested relations, by utilising the above concepts.

Definition 4.7. Let **R**(F) be a joinable NDS, over the joined NRS, U(T), and let r* be a nested relation over U(T). Then, the NEJD, $\bowtie^{ne}$[**R**(F)], holds in r* if and only if whenever there exist n not necessarily distinct tuples, $t_1, t_2, ..., t_n \in$ r*, that are combinable with the resulting tuple, t*, over U(T), then t* ∈ r* also holds.

We say that a nested relation r*, over U(T), possesses a *null extended lossless decomposition* onto **R**(F) if and only if r* satisfies the NEJD, $\bowtie^{ne}$[**R**(F)].

We further note that a null extended lossless decomposition of r* onto a joinable NDS, **R**(F), does *not*, in general, imply that

$$r^* \cong \bowtie^{ne}\ {}_{i=1}^{q}\ (\Pi^{ne}{}_{S(T_i)}\ (r^*))$$

holds as is the case in flat relational database theory without null values. This is due to the inequality rule for nulls (see Definition 2.5), whereby *nulls* in projected attribute values are not equal. We note that in [LEVE90a] we show that in the special case when FDS(F) is γ-*acyclic* [FAGI83] then the *null extended outer join* defined in [LEVE91] can be utilised to join all the nested relations in a nested database d* without loss of information.

If q = 1, then $\bowtie^{ne}[\mathbf{R}(F)]$ is said to be a *trivial* NEJD, otherwise it is said to be a *non-trivial* NEJD. If q = 2, i.e. $\mathbf{R}(F) = \{R(T_1), R(T_2)\}$, then $\bowtie^{ne}[\mathbf{R}(F)]$ is a *null extended multivalued dependency* (NEMVD).

The next theorem shows that the NEJD is faithful to the NJD.

THEOREM 4.1. *Let* **R** *be a FDS, over U, and let r be a flat relation over U. Then, the NEJD,* $\bowtie^{ne}[\mathbf{R}]$*, is faithful to the NJD,* $\bowtie[\mathbf{R}]$*, in the sense that* $\bowtie^{ne}[\mathbf{R}]$ *holds in r if and only if* $\bowtie[\mathbf{R}]$ *holds in r.*

PROOF. The result follows, since in this case Definition 4.7 of the NEJD reduces to Definition 4.6 of the NJD by the following argument.

Let $t_1, t_2, ..., t_n \in$ r be *combinable* on a covering subset FDS, **S**, of **R**, i.e. $\mathbf{S} \in$ SUBSET(**R**), where (n = |**S**|) ≤ (q = |**R**|), with the resulting tuple, t, over U. Thus, by Definition 4.6, $t[S_i] \cong t_i[S_i]$, $S_i \in \mathbf{S}$, $1 \leq i \leq n$, and t[MANY(**S**)] is MANY(**S**)-total.

Equivalently, we have that

$$t \cong \bowtie_{i=1}^{n} (\Pi_{S_i}(t_i)) \neq \emptyset,$$

where $S_i \in \mathbf{S}$, $1 \leq i \leq n$, and t[MANY(**S**)] is MANY(**S**)-total.

Now, by Theorem 3.2, Π^{ne} is faithful to Π, and, by Theorem 3.4, $\bowtie^{ne}$ is faithful to $\bowtie$. The result thus follows, since, by Definition 4.7, there exists a tuple, t, over U, such that

$$t \cong \bowtie^{ne}{}_{i=1}^{n} (\Pi^{ne}{}_{S_i}(t_i)) \neq \emptyset,$$

where $S_i \in \mathbf{S}$, $1 \leq i \leq n$, and t[MANY(**S**)] is MANY(**S**)-total. □

Theorem 4.1 states that the effect of null values on NJDs (or on NMVDs) is the same as the effect of null values on NEJDs (or on NEMVDs), when we consider only flat relations over FDSs.

Example 4.3. Let r be the flat relation, shown in Figure 4.2. Then, it can easily be verified that r satisfies the NMVD, A $\rightarrow\rightarrow$ B | C (or equivalently $\bowtie$[{AB, BC}]), and correspondingly r satisfies the NEMVD, A $\rightarrow\rightarrow$ B | C (or equivalently $\bowtie^{ne}$[{AB, AC}]).

A	B	C
null	b_1	c_1
null	b_2	c_2

Fig. 4.2. The flat relation with *null*, r, satisfying the NEMVD $\bowtie^{ne}$[{AB, AC}].

The next theorem shows that the NEJD is a precise generalisation of the NJD.

THEOREM 4.2. *Let* **R**(F) = $\{R(T_1), R(T_2), ..., R(T_q)\}$ *be the joinable NDS over the joined NRS, U(T), and let r* be a nested relation over U(T). Then, the NEJD,* $\bowtie^{ne}$[**R**(F)], *is a precise generalisation of the NJD,* $\bowtie$[*FDS(F)*], *in the sense that the NEJD,* $\bowtie^{ne}$[**R**(F)], *holds in r* if and only if the NJD,* $\bowtie$[*FDS(F)*], holds in μ*(r*).

PROOF. By Theorem 4.1, the NJD, $\bowtie$[FDS(F)], holds in μ*(r*) if and only if the NEJD, $\bowtie^{ne}$[FDS(F)], holds in μ*(r*). It, therefore, suffices to show that $\bowtie^{ne}$[**R**(F)] holds in r* if and only if $\bowtie^{ne}$[FDS(F)] holds in μ*(r*).

IF. The NEJD, $\bowtie^{ne}$[FDS(F)], holds in μ*(r*) if and only if whenever there exist n not necessarily distinct tuples, $t_1, t_2, ..., t_n \in$ μ*(r*), with n $\leq$ q, that are combinable on a covering subset FDS, FDS(F′), of FDS(F), i.e. FDS(F′) $\in$ SUBSET(FDS(F)), where |FDS(F′)| = n, with the resulting tuple, t, over U, then t $\in$ μ*(r*).

Let FDS(F′) = {S(T'_1), S(T'_2), ..., S(T'_n)}. Now, by (1) and the faithfulness of Π^{ne} and $\bowtie^{ne}$, established in Theorems 3.2 and 3.4, respectively, it follows that

$$t \cong \bowtie_{i=1}^{n} (\Pi_{S(T'_i)}(t_i)) \neq \emptyset. \tag{2}$$

It is also true that there exist n not necessarily distinct tuples, $t_i^* \in$ r*, where $t_i \in \mu^*(t_i^*)$, 1 $\leq$ i $\leq$ n. (In fact there may be more distinct t_i's than t_i^*'s.) It, therefore, follows from (2) that

$$t \in \bowtie_{i=1}^{n} (\Pi_{S(T'_i)} (\mu^*(t_i^*))) \neq \emptyset. \tag{3}$$

Thus, from (3) and the preciseness of Π^{ne} and $\bowtie^{ne}$, established in Theorems 3.2 and 3.4, respectively, we have that

$$t^* \cong \bowtie^{ne}{}_{i=1}^{n} (\Pi^{ne}{}_{S(T'_i)} (t_i^*)) \neq \emptyset, \tag{4}$$

with t $\in$ μ*(t*).

We now conclude the *if* part of the proof; t* $\in$ r* must obtain, otherwise the NEJD, $\bowtie^{ne}$[FDS(F)], would be violated in μ*(r*). Thus, by (4) and Definition 4.7, $\bowtie^{ne}$[**R**(F)] holds in r* as required.

ONLY IF. The NEJD, $\bowtie^{ne}$[**R**(F)], holds in r* if and only if whenever there exist n not necessarily distinct tuples, t_1^*, t_2^*, ..., t_n^* $\in$ r*, that are combinable on a covering subset NDS, **R**(F′), of **R**(F), i.e. **R**(F′) $\in$ SUBSET(**R**(F)), where |**R**(F′)| = n, with the resulting tuple, t*, over U(T), then t* $\in$ r*.

We prove the *only if* part by reversing the argument of the *if* part of the proof.

Let F′ = {T'_1, T'_2, ..., T'_n}. Then from (1) we obtain equation (4). Now, on using the preciseness of Π^{ne} and $\bowtie^{ne}$, established in Theorems 3.2 and 3.4, respectively, and the operator μ* we derive equation (3). Equation (3) asserts the existence of a tuple, t, over U, such that t $\in$ μ*(t*). It now follows that equation (2) describes the construction of the tuple t $\in$ μ*(t*).

We now conclude the *only if* part of the proof; t $\in$ μ*(r*) must obtain, otherwise the NEJD, $\bowtie^{ne}$[**R**(F)], would be violated in r*. Thus, by (2) and Definition 4.7, $\bowtie^{ne}$[FDS(F)] holds in μ*(r*) as required. □

Example 4.4. Let **R**(F) = {R(T_1), R(T_2)}, where R(T_1) = A(B)*(D)*, R(T_2) = Λ(B(C)*)* are projected NRSs of U(T) = A(B(C)*)*(D)*. It can easily be verified that the nested relation, r*, over U(T), shown in Figure 4.3, violates the NEJD, $\bowtie^{ne}$[**R**(F)]. On the other hand, it can also be verified that the nested relation, r′*, over U(T), shown in Figure 4.4, satisfies the given NEJD,

$\bowtie^{ne}$[**R**(F)].

A	(B	(C)*)*	(D)*
	B	(C)*	D
		C	
a_1	b_1	c_1	d_1
	b_2	c_1	d_2
a_2	b_1	c_2	d_1
	b_3	c_2	d_3

Fig. 4.3 The nested relation, r*, which violates $\bowtie^{ne}$[{A(B)*(D)*, Λ(B(C)*)*}].

A	(B	(C)*)*	(D)*
	B	(C)*	D
		C	
a_1	b_1	c_1	d_1
	b_2	c_1	d_2
a_1	b_1	c_2	d_1
			d_2
a_2	b_1	c_2	d_1
	b_3	c_2	d_3
a_2	b_1	c_1	d_1
			d_3

Fig. 4.4. The nested relation, r′*, which satisfies $\bowtie^{ne}$[{A(B)*(D)*, Λ(B(C)*)*}].

We observe that, by Theorem 4.2, it can be easily shown that the set of inference rules in Lien [LIEN79, LIEN82], shown therein to be sound and complete for NFDs and NMVDs, is also sound and complete for NFDs and NEMVDs. In the more general case, when we consider NEJDs instead of just NEMVDs, the special case when only one NEJD, over U(T), is considered (which corresponds to a single NJD over U) is interesting. This is because it has been recently shown by Miller et al. [MILL88] that the set of inference rules given by Sciore [SCIO82] is sound and complete for inferring all the JDs that are implied by a single JD over U.

We close this subsection with the definition of the NEJD-rule for the NEJD, $\bowtie^{ne}$[**R**(F)], with respect to a nested relation, r*, over the joined NRS, U(T).

Definition 4.8. Let **R**(F) be a joinable NDS over the joined NRS, U(T); let $\bowtie^{ne}[\mathbf{R}(F)]$ be an NEJD, over U(T), and let r* be a nested relation over U(T). Then the *NEJD-rule* for the NEJD, $\bowtie^{ne}[\mathbf{R}(F)]$, denoted as $RULE_{\bowtie^{ne}[\mathbf{R}(F)]}(r^*)$, is defined as follows:

Let $t_1, t_2, ..., t_n \in$ r* be n not necessarily distinct tuples that are combinable with the resulting tuple, t*, over U(T). If there is no tuple $t^{*\prime} \in r^*$ such that $t^{*\prime} \geq t^*$, then $RULE_{\bowtie^{ne}[\mathbf{R}(F)]}(r^*) \equiv r'^*$, where r'^* is r* after the assignment $r'^* := r^* \cup^{ne} \{t^*\}$.

(Cf. [GRAH86, MAIE79, MEND84, STEI85, SAGI83, SAGI88].)

4.4. Null Hierarchical Dependencies

In this subsection we define the *null hierarchical dependency*, which characterises the set of NEFDs, FD(T), that are naturally represented within a single scheme tree, T. Our main result in this section, i.e. Theorem 4.3, presents equivalent characterisations of the subclass of nested relations satisfying a null hierarchical dependency. In particular, for the subclass of nested relations, called *nested flat relations* (Definition 4.9), FD(T) is satisfied in a nested relation, r*, over R(T), if and only if the set of NMVDs embedded in the scheme tree, T, denoted by MVD(T), is satisfied in the flat relation, $\mu^*(r^*)$. In addition, MVD(T) is satisfied in $\mu^*(r^*)$ if and only if the NJD represented by the path set of T, i.e $\bowtie[P(T)]$, is satisfied in $\mu^*(r^*)$. This result generalises Proposition 4.1 from [OZSO87], wherein it was shown that MVD(T) $\approx \bowtie[P(T)]$ for nested relations without *nulls*, and Theorem 7 from [FISC85], wherein it was shown that a MVD, $X \rightarrow\rightarrow Y$, holds in a nested relation (without *nulls*), r*, over R(T), if and only if the NEFD, $X \rightarrow (Y)^*$, holds in the nested relation, $\nu_Y(r^*)$, resulting from a NEST operation over $Y \in R(T)$, where $X \cap Y = \emptyset$.

We now define a subclass of nested relations which have the property that they can be obtained from their flat counterparts by using only NEST operations.

Definition 4.9. A *nested flat relation*, r*, over a NRS, R(T), is a nested relation that can be obtained from the flat relation, $\mu^*(r^*)$, over S(T), by using a sequence of NEST operations. In other words, let r be a flat relation over S(T) $\subseteq$ U, then, for an integer n $\geq$ 0,

$$r^* \equiv \nu_{Y_n}(\nu_{Y_{n-1}} \ ... \ (\nu_{Y_1}(r)) \ ...)$$

is a nested relation over the NRS, R(T) = R(T_n), with $Y_i \in$ R(T_{i-1}), 1 $\leq$ i $\leq$ n, where R(T_0) = S(T) and R(T_i) = (R(T_{i-1}) - Y_i)(Y_i)*, i > 0, is the NRS over which $\nu_{Y_i}(\nu_{Y_{i-1}}$ ... $(\nu_{Y_1}$(r)) ...) is defined.

Following Ozsoyoglu and Yuan [OZSO87] we next define the NMVD represented by an edge of T.

Definition 4.10. Let e = {u,v} be an edge in a scheme tree, T. Then we denote by MVD(e) the NMVD represented by the edge e, namely, A(u) $\rightarrow\rightarrow$ D(v) (S(T)).

We let MVD(T) denote the set of NMVDs represented by all the edges of T. We note that for nested relations without *nulls* the set of NMVDs, MVD(T), is equivalent to a set of MVDs which in turn are equivalent to a *generalised hierarchical decomposition* [DELO78].

The set of NMVDs represented by F, denoted by MVD(F), is given by MVD(F) = $\cup_{i=1}^{q}$ MVD(T_i).

Example 4.5. For the scheme tree, T_1, shown in Figure 2.1, we have:

MVD(e_1) = STUDENT,DEPT $\rightarrow\rightarrow$ MAJOR (S(T))

MVD(e_2) = STUDENT,DEPT $\rightarrow\rightarrow$ CLASS,EXAM,PROJECT (S(T))

MVD(e_3) = STUDENT,DEPT,CLASS $\rightarrow\rightarrow$ EXAM (S(T))

MVD(e_4) = STUDENT,DEPT,CLASS $\rightarrow\rightarrow$ PROJECT (S(T))

The set of NMVDs represented by the edges of T_1 is given by MVD(T_1) = {MVD(e_1), MVD(e_2), MVD(e_3), MVD(e_4)}.

It can be verified that the nested relation, r*, shown in Figure 2.2. is a hierarchical relation, i.e. r* is a nested flat relation and μ*(r*) satisfies its set of NMVDs, MVD(T_1). It can also be verified that P(T_1) is γ-acyclic [LEVE89b] and that MVD(T_1) $\approx$ $\bowtie$[P(T_1)] (cf. [OZSO87]) by the non-split substitution rule and thus $\mu^*(r^*)$ also satisfies $\bowtie[P(T_1)]$.

We next define the set of NEFDs represented by a scheme tree T; in analogy to MVD(T), we denote such a set by FD(T).

Definition 4.11. Let e = (u,v) be an edge in a scheme tree, T, and let the NMVD for the edge, e, be MVD(e) = A(u) $\rightarrow\rightarrow$ D(v) (S(T)). Let T_v be the subtree rooted at the node, v, of T. Then, the NEFD, corresponding to MVD(e), which holds in a nested relation r*, over R(T), is A(u) $\rightarrow$ (R(T_v))*, and is denoted as FD(e).

The set of NEFDs, which are represented by the edges of a scheme tree, T, is denoted by FD(T); we call FD(T) *a null hierarchical dependency*. The set of NEFDs, which are represented by the scheme forest, F = $\{T_1,T_2,...,T_q\}$, is denoted by FD(F) and is given by FD(F) = $\cup_{i=1}^{q}$ FD(T_i).

Example 4.6. For the scheme tree, T_1, of the running example shown in Figure 2.1 we have:

FD(e_1) = STUDENT,DEPT $\rightarrow$ (MAJOR)*

FD(e_2) = STUDENT,DEPT $\rightarrow$ (CLASS (EXAM)* (PROJECT)*)*

FD(e_3) = STUDENT,DEPT,CLASS $\rightarrow$ (EXAM)*

FD(e_4) = STUDENT,DEPT,CLASS $\rightarrow$ (PROJECT)*

For the scheme tree, T_2, of the running example shown in Figure 2.5 we have:

FD(e_5) = CLASS $\rightarrow$ (TUTOR)*

FD(e_6) = CLASS $\rightarrow$ (TEXT)*

For the scheme tree, T_3, of the running example shown in Figure 1.1 we have:

FD(e_7) = TUTOR,SALARY $\rightarrow$ (CHILD)*

FD(e_8) = TUTOR,SALARY $\rightarrow$ (DAY)*

It thus follows that FD(T_1) = {FD(e_1), FD(e_2), FD(e_3), FD(e_4)}, FD(T_2) = {FD(e_5), FD(e_6)} and FD(T_3) = {FD(e_7), FD(e_8)}. Finally, for the scheme forest, F, of the running example, we have FD(F) = FD(T_1) $\cup$ FD(T_2) $\cup$ FD(T_3).

It can easily be verified that FD(F) holds for the nested database d* = {r_1^*, r_2^*, r_3^*} of the running example, where r_1^* is shown in Figure 2.2, r_2^* is shown in Figure 2.6 and r_3^* is shown in Figure 1.2.

The following theorem establishes a relationship between FD(T), MVD(T) and $\bowtie$[P(T)].

THEOREM 4.3 [LEVE90a]. *The following statements are equivalent.*

(1) *The nested relation, r*, over R(T), is a nested flat relation that satisfies FD(T).*

(2) *The flat relation,* μ**(r*), over S(T), satisfies MVD(T).*

(3) *The flat relation,* μ**(r*), over S(T), satisfies* $\bowtie$[*P(T)*].

PROOF (Sketch). We first prove that (1) is equivalent to (2), i.e. FD(T) ≈ MVD(T). In order to prove this result, we show that (1) => (2) by induction on the number of UNNEST operations required to obtain μ*(r*) from r*. The inductive hypothesis assumes, in this case, that n applications of UNNEST to a given nested flat relation, say r′*, in which a set of NEFDs holds, will yield a set of NMVDs that hold in the resulting flat relation, μ*(r′*). We then prove that (2) => (1) by induction on the number of NEST operations required to obtain r* from μ*(r*). The inductive hypothesis assumes, in this case, that n applications of NEST to a given flat relation, say μ*(r′*), in which a set of NMVDs holds, will yield a set of NEFDs that hold in the nested flat relation, r′*.

To conclude the proof we prove that (2) is equivalent to (3), i.e. MVD(T) ≈ $\bowtie$[P(T)]. In order to prove this result, we show that (3) => (2) by using the *covering rule* for JDs [BEER81], which is also sound for NJDs. Finally, we show that (2) => (3) by using the *non-split substitution rule* established in Sunbsection 4.3. □

5. THE EXTENDED CHASE PROCEDURE

In this section we extend the chase procedure to nested relations, and call it the *extended chase* procedure; this procedure is used in order to test satisfaction of a set of null extended data dependencies, say D(U), and to infer more information from a given nested relation by using the null extended data dependencies in D(U).

We then define the meaning of applying the extended chase to a nested relation, r*, over a joined NRS, U(T), with respect to a set of *null extended data dependencies*, say D(U); we denote the result of the said application by $CHASE_{D(U)}$(r*). Informally, the meaning of $CHASE_{D(U)}$(r*) is the result of applying repetitively the rules associated with the null extended data dependencies in D(U) until no more rules can be applied, i.e. a fixpoint is attained. In the case when $CHASE_{D(U)}$(r*) ≡ {*null*}, i.e. a NFD in D(U) is violated, we say that r* is *inconsistent*, otherwise we say that r* is *consistent* (cf. [ATZE87, GRAH86, MAIE79, MEND84]).

Finally, we investigate the properties of $CHASE_{D(U)}$(r*). We show that $CHASE_{D(U)}$(r*) terminates and satisfies the set of null extended data dependencies, D(U), provided $CHASE_{D(U)}$(r*) is consistent. Moreover, on letting D be the set of *null data dependencies* emanating from D(U), i.e. the counterparts of the null extended data dependencies that hold in the corresponding flat relations, we show that $CHASE_{D(U)}$(r*) generalises $CHASE_D$(r), where r ≅ μ*(r*), in a very natural way. An immediate consequence of these results is that we can now characterise a null hierarchical dependency in terms of the extended chase, namely, a null hierarchical dependency, FD(T), holds in a nested flat relation, r*, over the NRS, R(T), if and only if

$$r^* \cong CHASE_{FD(T)}(r^*).$$

Hereafter, we denote the set of null extended data dependencies that hold in a nested relation, over U(T), by D(U). The set D(U) may include:

(1) NFDs in FF(T), represented in the nodes of the scheme tree, T;

(2) NEFDs in FD(T), represented by the edges of the scheme tree, T; and

(3) the NEJD, $\bowtie^{ne}[\mathbf{R}(F)]$, over the joined NRS, U(T).

Before we define the *extended chase* of a nested relation, r*, over U(T), with respect to D(U), we define the effect on r* of invoking all possible applications to r* of the rules associated with the null extended data dependencies in D(U). Thus, we let

$$RULE_{D(U)}(r^*) \equiv \{RULE_{d_i^*}(r^*) \mid d_i^* \in D(U)\},$$

i.e. we invoke all possible applications to r* of the rules associated with the null extended data dependencies $d_i^* \in D(U)$ in *parallel,* since no ordering is assumed when applying the above rules.

We now define a transformation function $T_{D(U)}$, which operates on a nested relation r*, as follows:

(1) If $null \in RULE_{D(U)}(r^*)$, then $T_{D(U)}(r^*) \equiv \{null\}$, i.e it is undefined; otherwise

(2) $T_{D(U)}(r^*) \equiv \hat{\{} \cup^{ne} \{r'^* \mid r'^* \in RULE_{D(U)}(r^*)\} \hat{\}}$.

(We recall that $\hat{\{}\, r^* \,\hat{\}}$ is the reduced nested relation corresponding to r*.)

We next define successive applications of $T_{D(U)}$ to r* by:

$$T^0_{D(U)}(r^*) \equiv r^*;$$

$$T^{i+1}_{D(U)}(r^*) \equiv \hat{\{}\, T_{D(U)}(T^i_{D(U)}(r^*)) \cup^{ne} T^i_{D(U)}(r^*) \,\hat{\}} \text{ with i = 0,1,2,....}$$

Finally, we define the *extended chase* of r* with respect to a set of null extended data dependencies, D(U), denoted by $CHASE_{D(U)}(r^*)$ (or simply CHASE(r*) when D(U) is understood from context), as the *fixpoint* of r* with respect to $T_{D(U)}$, namely

$$CHASE_{D(U)}(r^*) \equiv \hat{\{} \cup^{ne}{}_{i=0}^{\infty} T^i_{D(U)}(r^*) \hat{\}}.$$

If CHASE(r*) $\not\equiv \{null\}$ then we say that r* is *consistent* with respect to D(U) (or simply consistent, when D(U) is understood from context), otherwise we say that r* is *inconsistent* with respect to D(U) (or simply inconsistent, when D(U) is understood from context).

We note that CHASE is an *inflationary fixpoint operator* [GURE86, KOLA88] provided r* is consistent, since it can easily be verified that for any nested relation, r′*, over U(T), r′* $\sqsubseteq$ $T_{D(U)}(r'^*)$. The motivation for using inflationary semantics is that the uniqueness of the extended chase follows directly from its parallel semantics rather than by a laborious proof employing double induction as in [LEVE90a].

The following results, which are a generalisation of corresponding results in [MAIE79], obtain for a nested relation, r*, over U(T), and a set, D(U), of null extended data dependencies.

LEMMA 5.1. *CHASE(r*) is finite and unique.*

PROOF. We first prove that CHASE(r*) is finite. By the definition of CHASE(r*), if no further rules can be applied to r* that either modify the tuples of r* or incorporate more informative tuples into it, then CHASE(r*) terminates. Also, by the definition of CHASE(r*), if r* is inconsistent then CHASE(r*) also terminates. So we assume that an application of a rule transforms r*, say, to r′*. We need consider the following cases:

CASE 1. A NFD-rule is applied to r*. Then either *null* is replaced by a more informative value, in which case r* $\sqsubseteq$ r′*, or else r* violates the NFD associated with the NFD-rule and it is, therefore, inconsistent.

CASE 2. A NEFD-rule is applied to r* and therefore r* $\sqsubseteq$ r′*, since an information-wise equivalence is enforced in r*.

CASE 3. A NEJD-rule is applied to r* and therefore r* $\sqsubseteq$ r′*, since a more informative tuple is incorporated into r*.

Now, let ATOMIC(r*) be the finite set of all atomic domain values in μ*(r*), together with *null*. We note that none of the rules add any new atomic domain values to ATOMIC(r*). Thus, it suffices to show that CHASE(r*) does not loop forever.

In what follows we refer to a *state* of CHASE(r*) as an intermediate state thereof, between r* and CHASE(r*), during the computation of CHASE(r*).

In order to conclude the proof, let r_i^* be the state of CHASE(r*) after applying some rules to r*, and let r_j^* be the state of CHASE(r*) after applying some further rules to r_i^*. Now, each time a rule is applied during the computation of CHASE(r*), a more informative nested relation results from such an application and thus $r_i^* \sqsubseteq r_j^*$; in addition, it is also true that $\neg(r_i^* \cong r_j^*)$ unless we have $r_i^* \cong$ CHASE(r*). The result now follows, since $r_i^* \sqsubseteq r_j^*$ can only occur finitely many times due to the finiteness of ATOMIC(r*).

We conclude the proof by showing that CHASE(r*) is unique. We observe that if r* is inconsistent then CHASE(r*) $\equiv$ {*null*} in which case the result follows, so we assume that r* is consistent. The result now follows since $T_{D(U)}$(r*) is unique by the definition of $RULE_{D(U)}$(r*), and CHASE(r*) results from successively applying the transformation function $T_{D(U)}$ to the current state of CHASE(r*). □

LEMMA 5.2. *If r* is consistent then CHASE(r*) satisfies D(U).*

PROOF. We consider the satisfaction of the different null extended data dependencies that may be included in D(U) in turn.

CASE 1. CHASE(r*) satisfies the NFDs in D(U), since the NFD-rule detects any violation of an NFD in D(U) (see Definition 4.3), and by assumption r* is consistent.

CASE 2. CHASE(r*) satisfies the NEFDs in D(U), since the NEFD-rule detects any violation of an NEFD in D(U) (see Definition 4.5).

CASE 3. CHASE(r*) satisfies the NEJD in D(U), since the NEJD-rule detects any violation of the NEJD in D(U) (see Definition 4.8).

From the above, it follows that CHASE(r*) satisfies D(U), otherwise, by the above argument, we can apply one of the rules to CHASE(r*), thus leading to a contradiction. □

We are now ready to present the main theorem of this section, which shows that an extended chase of a nested relation, r*, with respect to a set of null extended data dependencies, D(U), is information-wise equivalent to an extended chase of μ*(r*) with respect to the counterpart set of null data dependencies, D.

THEOREM 5.3. *Let* **R**(F) *be a joinable NDS over the joined NRS, U(T), and let r* be a nested relation over U(T). Also, let D(U) = {FF(T), FD(T),* $\bowtie^{ne}[\mathbf{R}(F)]$*} be a set of null extended data dependencies over U(T), and let D = {FF(T), MVD(T),* $\bowtie[FDS(F)]$*} be the counterpart set of null data dependencies over U. Then,*

$CHASE_D(\mu^*(r^*)) \cong \mu^*(CHASE_{D(U)}(r^*))$.

PROOF. To prove the result we first make the following claim.

CLAIM 1. For each rule associated with a null extended data dependency, say $d_i^* \in$ D(U), that is applied to r*, we can apply the corresponding rule associated with the counterpart null data dependency, say $d_i \in$ D, to $\mu^*(r^*)$.

Let r′* be the nested relation obtained from r* after applying to r* some rules associated with null extended data dependencies in D(U). By repetitively utilising claim 1, we have that $\mu^*(r'^*)$ is the flat relation obtained from $\mu^*(r^*)$ after applying to $\mu^*(r^*)$ the corresponding rules associated with the counterpart null data dependencies in D.

Now, by Lemma 5.1, since both $CHASE_{D(U)}(r^*)$ and $CHASE_D(\mu^*(r^*))$ terminate, the result follows.

In order to prove claim 1, and thereby conclude the proof of the theorem, we consider the following three cases:

CASE 1. Let X $\rightarrow$ A be a NFD in D(U). Now, let t be the XA-value resulting from the modification of one of the tuples, $t_1, t_2 \in \mu^*(\Pi^{ne}{}_{XA}(r^*))$, as a consequence of the NFD-rule associated with the NFD, X $\rightarrow$ A. Then, the XA-value, t, also results from the modification of one of the tuples, $t_1, t_2 \in \Pi_{XA}(\mu^*(r^*))$, as a consequence of the NFD-rule associated with the NFD, X $\rightarrow$ A, since, by Theorem 3.2, Π^{ne} is a precise generalisation of Π. Thus, applying the NFD-rule associated with the NFD, X $\rightarrow$ A, to r*, corresponds to applying the NFD-rule associated with the NFD, X $\rightarrow$ A, to $\mu^*(r^*)$.

CASE 2. Let e = (u,v) be an edge in the scheme tree, T, associated with r*, and let T_v be the subtree of T rooted at the node v. Also, let X = A(u), Y = D(v) and (Y)* = (R(T_v))*. For simplicity, we assume that X = Z(R(T)) and (Y)* $\in$ H(R(T)). If this is not the case then we can repetitively unnest r*, corresponding to recursive applications of the NEFD-rule in Definition 4.5 (2), thus obtaining r′* over R(T′) such that X = Z(R(T′)) and (Y)* $\in$ H(R(T′)).

Now, let t* be a tuple resulting from the modification of not necessarily distinct tuples, t_1, t_2 $\in$ r*, by the application of the NEFD-rule associated with the NEFD, X $\rightarrow$ (Y)*. As a result, t*[X] = t_1[X] = t_2[X] and, in addition, t*[U(T) - X(Y)*] $\cong$ t_1[U(T) - X(Y)*] or t*[U(T) - X(Y)*] $\cong$ t_2[U(T) - X(Y)*]; moreover, t*[(Y)*] $\cong$ (t_1[(Y)*] $\cup^{ne}$ t_2[(Y)*]). Then, the additional tuples in μ*(t*) are incorporated into μ*(r*) by applying the NEJD-rule associated with the NMVD, X $\rightarrow\rightarrow$ Y, to the tuples in μ*(t_1) and μ*(t_2), since, by Theorem 3.1, $\cup^{ne}$ is a precise generalisation of $\cup$ and, by Theorem 4.2, the NEMVD is a precise generalisation of the NMVD. Thus, applying the NEFD-rule associated with the NEFD, X $\rightarrow$ (Y)*, to r*, corresponds to applying the NEJD-rule associated with the NEMVD, X $\rightarrow\rightarrow$ Y, to μ*(r*).

CASE 3. Let $\bowtie^{ne}$ be the NEJD in D(U). Now, let t* be a tuple that was incorporated into r* by an application of the NEJD-rule associated with the NEJD, $\bowtie^{ne}$[**R**(F)]. Then, the tuples in μ*(t*) are incorporated into μ*(r*) by applying the NEJD-rule associated with the NJD, $\bowtie$[FDS(F)], to μ*(r*), since, by Theorem 4.2, the NEJD is a precise generalisation of the NJD. Thus, applying the NEJD-rule associated with the NEJD, $\bowtie^{ne}$[**R**(F)], to r*, corresponds to applying the NEJD-rule associated with the NJD, $\bowtie$[FDS(F)], to μ*(r*). □

Our next corollary, which is a direct consequence of Theorem 5.3, tells us that testing for consistency in a nested relation, r*, is equivalent to testing for consistency in its flat counterpart, i.e. μ*(r*).

COROLLARY 5.4. *Let r*, D(U) and D be as in Theorem 5.3. Then, r* is consistent with respect to D(U) if and only if* μ**(r*) is consistent with respect to D.* □

We now give some examples in order to illustrate the extended chase procedure.

Example 5.1. Firstly, we give an example of an extended chase with NFDs and NEFDs. Consider the NRS, R(T) = A(BCD)*, together with D(U) = {FF(T), FD(T)}, where FF(T) = {B$\rightarrow$ C, C $\rightarrow$ D} and FD(T) = {A $\rightarrow$ (BCD)*}. Let r* be a nested relation over R(T), shown in Figure 5.1. Now, r* does not satisfy FF(T) or FD(T), so we extend chase r* with the NFD-rules associated with the NFDs in FF(T) and with the NEFD-rules associated with the NEFDs in FD(T) to obtain CHASE(r*), shown in Figure 5.2, which satisfies both FF(T) and FD(T).

A	(B	C	D)*
	B	C	D
a_1	b_1	*null*	d_1
a_1	b_2	*null*	*null*
a_2	b_1	c_1	*null*

Fig. 5.1. r* before the extended chase.

A	(B	C	D)*
	B	C	D
a_1	b_1	c_1	d_1
	b_2	*null*	*null*
a_2	b_1	c_1	d_1

Fig. 5.2. r* after the extended chase with respect to FF(T) and FD(T).

A	(B	(C)*)*	(D	E)*
	B	(C)*	D	E
		C		
a_1	b_1	c_1	d_1	*null*
	b_2	c_1	d_2	e_2
a_2	b_1	c_2	d_1	e_1
	b_3	c_2	d_3	*null*

Fig. 5.3. r* before the extended chase.

Example 5.2. Let r* be the nested relation, over R(T) = A(B(C)*)*(DE)*, shown in Figure 5.3. Let D(U) = {FF(T), FD(T), $\bowtie^{ne}$[A(B)*(DE)*,Λ(B(C)*)*]}, where FF(T) = {D $\rightarrow$ E} and FD(T) = {A $\rightarrow$ (B(C)*)*, A $\rightarrow$ (DE)*, AB $\rightarrow$ (C)*}. Now, r* does not satisfy the NEJD, $\bowtie^{ne}$[A(B)*(DE)*,Λ(B(C)*)*] (cf. Example 4.4), nor the NFD D $\rightarrow$ E. We now extend chase r* with the NEJD-rule associated with $\bowtie^{ne}$[A(B)*(DE)*,Λ(B(C)*)*], the NFD-rule associated with

D $\rightarrow$ E, and the NEFD-rules associated with FD(T). The result $CHASE_{D(U)}$(r*), shown in Figure 5.4, satisfies D(U). Moreover, $CHASE_D$(μ*(r*)) is shown in Figure 5.5, where D = {FF(T), MVD(T), $\bowtie$[ABDE, BC]} and MVD(T) = {A $\rightarrow\rightarrow$ BC, A $\rightarrow\rightarrow$ DE, AB $\rightarrow\rightarrow$ C}. It can easily be verified that the result of Theorem 5.3 holds, i.e. μ*($CHASE_{D(U)}$(r*)) $\cong$ $CHASE_D$(μ*(r*)).

<table>
<tr><th rowspan="3">A</th><th colspan="2">(B (C)*)*</th><th colspan="2">(D E)*</th></tr>
<tr><th rowspan="2">B</th><th>(C)*</th><th rowspan="2">D</th><th rowspan="2">E</th></tr>
<tr><th>C</th></tr>
<tr><td rowspan="3">a_1</td><td rowspan="2">b_1</td><td>c_1</td><td>d_1</td><td>e_1</td></tr>
<tr><td>c_2</td><td rowspan="2">d_2</td><td rowspan="2">e_2</td></tr>
<tr><td>b_2</td><td>c_1</td></tr>
<tr><td rowspan="3">a_2</td><td rowspan="2">b_1</td><td>c_1</td><td>d_1</td><td>e_1</td></tr>
<tr><td>c_2</td><td rowspan="2">d_3</td><td rowspan="2">null</td></tr>
<tr><td>b_3</td><td>c_2</td></tr>
</table>

Fig. 5.4. r* after the extended chase with respect to D(U).

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_1	b_1	c_2	d_1	e_1
a_1	b_1	c_2	d_2	e_2
a_1	b_2	c_1	d_1	e_1
a_1	b_2	c_1	d_2	e_2
a_2	b_1	c_1	d_1	e_1
a_2	b_1	c_1	d_3	*null*
a_2	b_1	c_2	d_1	e_1
a_2	b_1	c_2	d_3	*null*
a_2	b_3	c_2	d_1	e_1
a_2	b_3	c_2	d_3	*null*

Fig. 5.5. The flat relation, $CHASE_D$(μ*(r*)) $\cong$ μ*($CHASE_{D(U)}$(r*)).

Example 5.3. Let R(T) = A(B)*(C)*(D(E)*)*, and let r* be the nested relation, over R(T), shown in Figure 5.6. Let D(U) = {FD(T)}, where FD(T) = {A $\rightarrow$ (B)*, A $\rightarrow$ (C)*, A $\rightarrow$ (D(E)*)*, AD $\rightarrow$ (E)*}. By Theorem 4.3, r* does not satisfy FD(T), and it is not a nested flat relation. If we extend chase r* with the NEFD-rules associated with the NEFDs in FD(T), we get CHASE(r*), shown in Figure 5.7, which is a nested flat relation that satisfies FD(T).

A	(B)*	(C)*	(D	(E)*)*
	B	C	D	(E)*
				E
a_1	b_1	c_1	d_1	*null*
	b_2	c_2	d_2	e_1
	b_3		d_2	e_2
a_1	b_1	*null*	d_1	e_1
	b_2		d_2	*null*

Fig. 5.6. r* before the extended chase.

A	(B)*	(C)*	(D	(E)*)*
	B	C	D	(E)*
				E
a_1	b_1	c_1	d_1	e_1
	b_2	c_2	d_2	e_1
	b_3			e_2

Fig. 5.7. r* after the extended chase with respect to FD(T).

We note that the extended chase can also be used to solve the implication problem for null extended data dependencies (cf. [GRAH86, MAIE79, MAIE84]). For example, it can be used to test whether a NDS possesses the desirable property of null extended lossless decomposition. Finally, the extended chase can also be used to investigate other desirable properties that may be present in a nested relation such as the commuting of a sequence of NEST operations or the minimality of its number of tuples with respect to its corresponding flat relation (cf. [MIUR86, TAKE89, VANG88]). Finally, we observe that the notion of a *rule* can be generalised and extended leading to a logic-based language for nested relations similar to the *Datalog* language for flat relations [SAGI88, ULLM88].

6. CONCLUSIONS

A semantics for null extended nested relations in terms of null extended data dependencies was presented together with the extended chase procedure for testing and enforcing the satisfaction of these null extended data dependencies in a nested relation. We defined the structure of a nested relation in terms of a preorder which generalises the Hoare ordering on powerdomains and a generic null value, *null*. This led us to define information-wise equivalent nested relations, allowing us to measure the relative information content of tuples and nested relations.

The null extended algebra was introduced incorporating *null* into the definitions of the null extended operators and it was shown that the three operators of the null extended algebra employed herein are faithful and precise representations of their flat counterparts. We defined the new class of null extended data dependencies that hold in nested relations and its counterpart, i.e. the class of null data dependencies that hold in the corresponding flat relations. Then we obtained a series of interesting results concerning the said data dependencies culminating in Theorem 4.3, which presents equivalent charactesisations of the subclass of nested relations satisfying a null hierarchical dependency. In addition, we have extended the chase procedure to nested relations, showing that the extended chase possesses all of the desirable properties of the standard chase, and by utilising the extended chase we were able to derive a new characterisation of consistent nested relations (see Corollary 5.4). Thus, the extended chase may provide a basis for developing an integrity constraint checker for nested relations.

As a result of the semantics we have attached to nested relations we are able to define the *nested universal relation model* (nested UR model) [LEVE90a], which extends the classical UR model [ATZE87, FAGI82, GRAH86, HONE82, MAIE84, MEND84, SAGI83, SAGI88, STEI85] to nested relations. This is achieved by extending the *weak instance* approach to the UR model [ATZE87, HONE82, GRAH86, MAIE84, SAGI83] to the *nested weak instance* approach to the nested UR model. The central data structure of the nested UR model is the *nested representative instance*, which extends the representative instance of the UR model [ATZE87, HONE82, GRAH86, MAIE84, SAGI83] to nested relations. Informally, the nested representative instance is constructed by padding the nested relations in the database with *nulls* and then applying the extended chase to the null extended union of these padded nested relations with respect to a set D(U) of null extended data dependencies. The classical UR model has played one of the central roles in flat relational database theory and thus we think that the nested UR model is of major importance in nested relational database theory, since flat relations are just special cases of nested relations. We will formally present the nested UR model in a follow-up paper.

ACKNOWLEDGEMENTS

We would like to thank the anonymous referees for their helpful comments which improved the paper.

REFERENCES

[ABIT86] ABITEBOUL, S., AND BIDOIT, N. Non first normal form relations: An algebra allowing data restructuring. *J. Comput. Syst. Sci. 33*, 3 (1986), 361-393.

[ABIT89] ABITEBOUL, S., FISCHER, P. C., AND SCHEK, H.-J., Eds. *Nested Relations and Complex Objects in Databases*. Lecture Notes in Computer Science, vol. 361. Springer-Verlag, Berlin, 1989.

[ATZE86] ATZENI, P., AND MORFUNI, N. M. Functional dependencies and constraints on null values in database relations. *Information and Control 70*, (1986), 1-31.

[ATZE87] ATZENI P., AND BERNARDIS, M. C. A new basis for the weak instance model. In *Proceedings of the 6th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (San Diego, Calif.). ACM, New York, 1987, pp. 79-86.

[BEER81] BEERI, C., AND VARDI, M. Y. On the properties of join dependencies. In *Advances in Database Theory*, Vol. 1, H., Gallaire, J., Minker and J. M., Nicholas, Eds. Plenum Press, New York, 1981, pp. 25-72.

[BIDO87] BIDOIT, N. The Verso algebra or how to answer queries with fewer joins. *J. Comput. Syst. Sci. 35*, 3 (Dec. 1987), 321-364.

[BUNE91] BUNEMAN, P., JUNG, A., AND OHORI, A. Using powerdomains to generalize relational databases. *Theoret. Comput. Sci.*, to appear.

[CODD79] CODD, E. F. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 397-434.

[CODD86] CODD, E. F. Missing information (applicable and inapplicable) in relational databases. *ACM SIGMOD Record 15*, 4 (Dec. 1986), 53-78.

[COLB89] COLBY, L. S. A recursive algebra and query optimization for nested relations. In *Proceedings of the 1989 ACM-SIGMOD Conference on Management of Data*. ACM, New York, 1989, pp. 273-283.

[DELO78] DELOBEL, C. Normalization and hierarchical dependencies in the relational data model. *ACM Trans. Database Syst. 3*, 3 (Sept. 1978), 201-222.

[FAGI82] FAGIN, R., MENDELZON, A. O., AND ULLMAN, J. D. A simplified universal relation assumption and its properties. *ACM Trans. Database Syst. 7*, 3 (Sept. 1982), 343-360.

[FAGI83] FAGIN, R. Degrees of acyclicity for hypergraphs and relational database systems. *J. ACM 30*, 3 (July 1983), 514-550.

[FISC85] FISCHER, P. C., SAXTON, L. V., THOMAS, S. J., AND VAN GUCHT, D. Interactions between dependencies and nested relational structures. *J. Comput. Syst. Sci. 31*, 3 (Dec. 1985), 343-354.

[GRAH86] GRAHAM, M. H., MENDELZON, A. O., AND VARDI, M. Y. Notions of dependency satisfaction. *J. ACM 33*, 1 (Jan. 1986), 105-129.

[GURE86] GUREVICH, Y., AND SELAH, S. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic 32*, (1986), 265-280.

[GYSS89] GYSSENS, M., PAREDAENS, J., AND VAN GUCHT, D. A uniform approach toward handling atomic and structured information in the nested relational database model. *J. ACM 36*, 4 (1989), 790-825.

[HARA69] HARARY, F. *Graph Theory*. Addison-Wesley, Reading, Ma., 1969.

[HONE82] HONEYMAN, P. Testing satisfaction of functional dependencies. *J. ACM 29*, 3 (July 1982), 668-677.

[IMIE84] IMIELINSKI, T.. AND LIPSKI JR., W. Incomplete information in relational databases. *J. ACM 31*, 4 (Oct. 1984), 761-791.

[JAES82] JAESCHKE, G., AND SCHEK, H.-J. Remarks on the algebra of non first normal form relations. In *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (Los Angeles, Calif.). ACM, New York, 1982, pp. 124-138.

[JAJO90] JAJODIA, S., AND SPRINGSTEEL, F. N. Lossless outer joins with incomplete information. *BIT 30*, (1990), 34-41.

[KOLA88] KOLAITIS, P. G., AND PAPADIMITRIOU, C.H. Why not negation by fixpoint? *Proceedings of ACM Symposium on Principles of Database Systems* (Austin, Texas). ACM, New York, 1988, pp. 231-239.

[KORT88] KORTH, H. F. Optimization of object-retrieval queries. In *Proceedings of the 2nd International Workshop on Object-Oriented Database Systems* (Bad Munster am Stein-Ebernburg, West Germany). Lecture Notes in Computer Science, vol. 334. Springer-Verlag, Berlin, 1988, pp. 352-357.

[LERA86] LERAT, N., AND LIPSKI JR., W. Nonapplicable nulls. *Theoret. Comput. Sci. 46* (1986), 67-82.

[LEVE89a] LEVENE, M., AND LOIZOU, G. Modelling incomplete information in complex objects. In *Proceedings of the 7th British National Conference on Databases* (Edinburgh, Scotland). Cambridge University Press, Cambridge, 1989, pp. 241-259.

[LEVE89b] LEVENE, M., AND LOIZOU, G. γ-acyclic database schemes and nested relations. In [ABIT89], 1989, pp. 313-323.

[LEVE90a] LEVENE, M. The nested universal relation database model. PhD. thesis, Department of Computer Science, Birkbeck College, University of London, 1990.

[LEVE90b] LEVENE, M., AND LOIZOU, G. The nested relation type model: An application of domain theory to databases. *The Computer Journal 33*, 1 (1990), 19-30.

[LEVE91] LEVENE, M., AND LOIZOU, G. A Fully Precise Null Extended Nested Relational Algebra. Research Note RN/91/13, Department of Computer Science, University College London. Also, submitted for publication.

[LIEN79] LIEN, Y. E. Multivalued dependencies with null values in relational databases. In *Proceedings of 5th International Conference on Very Large Data Bases*, Rio de Janerio, Brazil, 1979, pp. 61-66.

[LIEN82] LIEN, Y. E. On the equivalence of database models. *J.ACM 29*, 2 (1982), 333-362.

[LIPS79] LIPSKI JR., W. 1979. On semantic issues connected with incomplete information databases. *ACM Trans. Database Syst. 4*, 3 (Sept. 1979), 262-296.

[MAIE79] MAIER, D., MENDELZON, A. O., AND SAGIV, Y. Testing implication of data dependencies. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 455-469.

[MAIE83] MAIER, D. *The Theory of Relational Databases*. Computer Science Press, Rockville, Maryland, 1983.

[MAIE84] MAIER, D., ULLMAN, J. D., AND VARDI, M. Y. On the foundations of the universal relation model. *ACM Trans. Database Syst. 9*, 2 (June 1984), 283-308.

[MAKI77] MAKINOUCHI, A. A consideration on normal form of not-necessarily-normalized relation in the relational data model. In *Proceedings of 3rd International Conference on Very Large Data Bases*, Tokyo, Japan, 1977, pp. 447-453.

[MEND84] MENDELZON, A. O. Database states and their tableaux. *ACM Trans. Database Syst. 9*, 2 (June 1984), 264-282.

[MILL88] MILLER, L. L., GADIA, S. K., KOTHARI, S., AND LIU, K. C. Completeness issues for join dependencies derived from the universal relation join dependency. *Inf. Process. Lett. 28*, (Aug. 1988), 269-274.

[MIUR86] MIURA, T., MORIYA, K., AND ARISAWA, H. Normalizing non first normal form relations. In *Proceedings of 6th Advanced Database Symposium*, Information Processing Society of Japan, 1986, pp. 65-71.

[OZSO87] OZSOYOGLU, Z. M., AND YUAN, L.-Y. A new normal form for nested relations. *ACM Trans. Database Syst. 12*, 1 (March 1987), 111-136.

[OZSO89] OZSOYOGLU, Z. M., AND YUAN, L.-Y. On the normalization in nested relational databases. In [ABIT89], 1989, pp. 243-271.

[ROTH88] ROTH, M. A., KORTH, H. F., AND SILBERSCHATZ, A. Extended algebra and calculus for nested relational databases. *ACM Trans. Database Syst. 13*, 4 (Dec. 1988), 389-417.

[ROTH89] ROTH, M. A., KORTH, H. F., AND SILBERSCHATZ, A. Null values in nested relational databases. *Acta Informatica 26*, (1989), 615-642.

[SAGI83] SAGIV, Y. A characterization of globally consistent databases and their correct access paths. *ACM Trans. Database Syst. 8*, 2 (June 1983), 266-286.

[SAGI88] SAGIV, Y. On bounded database schemes and bounded horn-clause programs. *SIAM J. Comput. 12*, (1988), 1-22.

[SCHE86] SCHEK, H.-J., AND SCHOLL, M. H. The relational model with relation-valued attributes. *Inf. Syst. 11*, 2 (1986), 137-147.

[SCHM86] SCHMIDT, D. A. *Denotational Semantics: A Methodology for Language Development*. Allyn and Bacon, Inc., Newton, Ma., 1986.

[SCIO82] SCIORE, E. A complete axiomatization of join dependencies. *J. ACM 29*, (1982), 373-393.

[STEI85] STEIN, J., AND MAIER, D. Relaxing the universal relation scheme assumption. In *Proceedings of the 4th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (Portland, Ore.). ACM, New York, 1985, pp. 76-85.

[TAKE89] TAKEDA, K. On the uniqueness of nested relations. In [ABIT89], 1989, pp. 139-150.

[THOM86] THOMAS, S. J., AND FISCHER, P. C. Nested relational structures. In *Advances in Computing Research*, Vol. 3, P. C., Kanellakis and F., Preparata, Eds. JAI Press, Greenwich, 1986, pp. 269-307.

[ULLM88] ULLMAN, J. D. *Principles of Database and Knowledge-Base Systems*, Vol. 1. Computer Science Press, Rockville, Maryland, 1988.

[VANG88] VAN GUCHT, D., AND FISCHER, P. C. Multilevel nested relational structures. *J. Comput. Syst. Sci. 36*, 1 (Feb. 1988), 77-105.

[ZANI84] ZANIOLO, C. Database relations with null values. *J. Comput. Syst. Sci. 28*, 1 (Feb. 1984), 142-166.