The Nested Universal Relation Data Model

MARK LEVENE
Department of Computer Science,
University College London, University of London,
Gower Street, London WC1E 6BT, U.K.,
email: mlevene@cs.ucl.ac.uk
AND
GEORGE LOIZOU
Department of Computer Science,
Birkbeck College, University of London,
Malet Street, London WC1E 7HX, U.K.,
email: george@cs.bbk.ac.uk

The Nested UR Model

List of Symbols

$\rightarrow$	arrow
$\emptyset$	empty set
Λ	capital greek lambda
ν	small greek new
μ	small greek mew
$\cup$	set union
$\cap$	set intersection
$\subseteq$	subset
$\times$	Cartesian product
$\in$	set membership
$\sqsubseteq$	special symbol
$\cong$	equal and ˜ on top
R	bold R
S	bold S
$\bowtie$	natural join operator
Π	capital greek pi
$\neg$	logical negation
$\downarrow$	vertical arrow pointing downwards
$\exists$	there exists
$\forall$	for all
$\wedge$	logical and
$\vee$	logical or
	close up p.18 and p.27

P.S. All letters to be italicized are clearly indicated in the text.

The nested universal relation (UR) model aims to provide *logical data independence* to the nested relational model by allowing users to view the database as if it were composed of a single nested relation. Moreover, non-technical users may find the nested relational model too complex to interact with, a problem we call herein the *usability problem*. The nested UR model solves the usability problem by allowing users to interact with the nested database without having to know its detailed structure, which may be complex.

In order to formalise the nested UR model we extend the weak instance approach to the (classical) UR model to the *nested weak instance approach* to the nested UR model. The nested weak instance approach leads naturally to the definition of the underlying data structure for the nested UR model, namely the *nested representative instance* (NRI) over the *nested universal relation scheme*. We present two different definitions of the NRI and show that they are equivalent. Firstly, we define the NRI declaratively as the *greatest lower bound* of the set of nested weak instances with respect to a natural ordering defined on nested relations. Secondly, we define the NRI constructively as the result of computing the *extended chase* on the underlying nested database. Finally, we show that the weak instance approach to the UR model is a special case of the nested weak instance approach to the nested UR model, thus allowing us to implement a flexible UR interface by using the nested UR model; this provides us with all the advantages of nested relations over flat relations.

1. INTRODUCTION

A database model provides *logical data independence* if changing the database at the conceptual level does not affect the user's view of the database. The classical *universal relation model* (UR model) [3, 10, 24, 25, 26, 27, 30, 32, 35] endeavours to achieve logical data independence in the *flat relational model* [8, 33] by allowing the user to view the database as if it were composed of a single flat relation. To this end, the user is provided with a UR interface [32] - with all the semantics embedded into the attributes - encapsulating the user's view of the database at the external level, on top of the conceptual level. The theory of the UR model was firmly established in the mid 1980's with the introduction of the *weak instance* approach [3, 13, 15, 24, 26]. In the weak instance approach to the UR model, the *representative instance* (RI) [24, 26, 27] becomes the underlying data structure of the UR model, which is suitable for storing all the data in the database in a single relation.

The *nested relational model* [1, 17, 28, 34] was developed in order to extend the applicability of the flat relational model to more complex non-business applications. Nested relations do not necessarily conform to the first normal form assumption of the flat relational model [8], thus allowing hierarchically structured complex objects to be modelled. The main advantages of nested relations in comparison to flat relations are: they minimise redundancy of data, and allow efficient query processing since some of the joins are realised within the nested relations themselves; in addition, nested relations allow explicit representation of the semantics of the application within their structures, and provide a more flexible user interface, which allows both flat and hierarchical data to be presented to the user.

One of the problems with the nested relational model is that it may prove too complex for non-technical users to interact with. This *usability problem* arises due to the fact that queries posed to a nested database may involve navigation both amongst and within the structure of nested relations in the nested database. Thus, as in the relational model, the nested relational model does not provide logical data independence. Moreover, posing queries to the nested database is much more difficult in the nested relational model than in the flat relational model due to the hierarchical structure of nested relations. The usability problem is accentuated even further when we take into account the application programs which may be impaired because of changes to the nested database at the conceptual level. In this paper we propose to alleviate the usability problem by providing logical data independence to the nested relational model. To this end we extend the (classical) UR model to nested relations by defining the *nested universal relation model* (nested UR model). In particular, we extend the weak instance approach to the UR model to the *nested weak instance* approach to the nested UR model.

EXAMPLE 1.1. Schemas of nested relations are represented graphically by *scheme trees* [17], such as T shown in Fig. 1.1. The *nested relation scheme* (NRS) of T, denoted by R(T), is:

AIRLINE AIR_CODE (FLIGHT_NO (PASSENGER)* (CREW)*)* (AIRPORT PORT_CODE)*,

where the higher order attributes are marked with * in order to distinguish them from the zero order attributes [1].

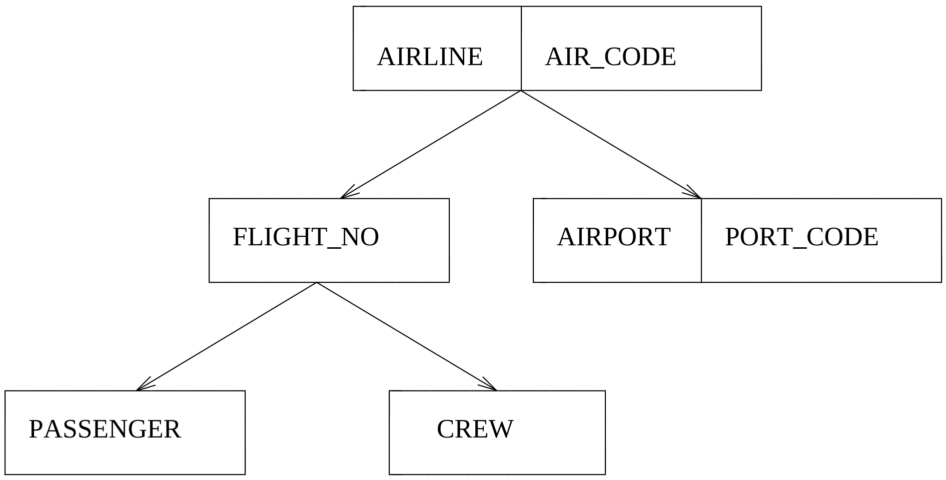

FIG. 1.1. The scheme tree T.

A null extended nested relation (abbreviated to nested relation), r*, over the NRS, R(T), for the scheme tree, T, of Fig. 1.1, is shown in Fig. 1.2. We note that *null* denotes a *generic null value*, whose semantics are represented by the partial ordering, *null* is *less informative than* any non-null value [17]. The semantics of r* can be expressed by a set of *null functional dependencies* (NFDs) and a set of *null extended functional dependencies* (NEFDs), both of which are members of the class of *null extended data dependencies* [17]. The NFDs that are satisfied in r* are: AIRLINE → AIR_CODE, AIR_CODE → AIRLINE, AIRPORT → PORT_CODE and PORT_CODE → AIRPORT. That is, an AIRLINE is associated with a unique AIR_CODE and vice versa. Similarly, an AIRPORT is associated with a unique PORT_CODE and vice versa. Furthermore, the NEFDs that are satisfied in r* are: AIRLINE,AIR_CODE → (AIRPORT,PORT_CODE)*, AIRLINE,AIR_CODE → (FLIGHT_NO (PASSENGER)* (CREW)*)*, AIRLINE,AIR_CODE,FLIGHT_NO → (PASSENGER)* and AIRLINE,AIR_CODE,FLIGHT_NO → (CREW)*. That is, every AIRLINE, AIR_CODE pair has a unique set of AIRPORT, PORT_CODE pairs, and a unique set of FLIGHT_NO (PASSENGER)* (CREW)* tuples. In addition, every AIRPORT, AIR_CODE, FLIGHT_NO triple has a unique set of PASSENGERs and CREWs.

The underlying data structure of the nested UR model is the *nested representative instance* (NRI) over the *nested universal relation scheme* (NURS). The NURS is a NRS, denoted by U(T), which allows us to model the semantics of a *nested database scheme* (i.e. a set of NRSs) within a single scheme tree, T, and whose attribute set is the universal set of attributes, U. The NRI, which extends the RI to nested relations, allows us to model the semantics of a nested database via an associated set of null extended data dependencies, D(U), within a single nested relation over the NURS, U(T). Thus, the NRI encapsulates all the information in the nested database within a single nested relation satisfying D(U).

AIRLINE	AIR_CODE	(FLIGHT_NO (PASSENGER)* (CREW)*)*			(AIRPORT	PORT_CODE)*
		FLIGHT_NO	(PASSENGER)*	(CREW)*	AIRPORT	PORT_CODE
			PASSENGER	CREW		
British-Airways	BA	213	Iris	Brian	Gatwick	LGW
			Mark	Anette	Heathrow	LHR
		214	*null*	Hanna	Ben-Gurion	*null*
Italian-Airways	AL-ITALIA	312	Mark	Robert	Orly	Paris
			Dan		NY	*null*
Israeli-Airways	EL-AL	213	*null*	*null*	Tel-Aviv	*null*
null	TWA	*null*	*null*	Reuven	Rome	*null*
				Naomi	*null*	*null*
		007	Mark	*null*	Heathrow	LHR
Dan-Air	*null*	707	Sara	*null*	Amsterdam	*null*
			Robert	Richard	*null*	FBI
		null	*null*	*null*	*null*	CIA
		700	Reuven	Mark	Rome	*null*
null	*null*	001	*null*	*null*	NY	*null*

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

We now briefly summarise the main results of the paper. Let d* be a nested database and D(U) be a set of null extended data dependencies. We present two different definitions of the NRI under D(U) for d* and show their equivalence. Firstly, we define the NRI under D(U) for d*, declaratively, as the *greatest lower bound* (GLB) [9] of the set of nested weak instances under D(U) for d*, which we denote by NWI(D(U), d*). We then show that NWI(D(U), d*) is a *complete semi-lattice* (called an *intersection structure* in [9]), i.e. that the GLB exists for any non-empty subset of NWI(D(U), d*) with respect to the ordering of *less informative than*, defined for nested relations. Secondly, we define the NRI under D(U) for d*, constructively, as the result of computing the *extended chase* [17] of d* padded with *nulls* to conform with the NURS, U(T). It follows that the NRI under D(U) for d* exists whenever NWI(D(U), d*) is non-empty or, equivalently, whenever d* is *consistent* (which informally means that it satisfies the set of NFDs in D(U)). The equivalence of the declarative and constructive definitions of the NRI implies that the extended chase is an effective tool for constructing the NRI. In addition, this equivalence generalises Theorem 1 in [24], where it was shown that the RI always produces the intersection of all weak instances. Furthermore, we show that the NRI naturally extends the RI in the sense that the NRI under D(U) for d* is shown to be equivalent to the RI under D for d; D is the set of *null data dependencies* [17], corresponding to D(U), that hold in the flat database, d, which is the result of completely unnesting all the nested relations in d*. Thus, the NRI maintains all the advantages of nested relations over flat relations and the RI becomes a special case of the NRI, i.e. when the nested database is a flat database.

The rest of the paper is organised as follows. In Section 2, we present the null extended nested relational model (abbreviated to nested relational model). In Section 3, we formalise the nested weak instance approach to the nested UR model and present our main results. Finally, in Section 4, we give our conclusions. The paper concludes with two appendices: in Appendix 1 we define the operators of the *null extended nested relational algebra* [19] used in the paper and in Appendix 2 we define the *null extended join dependency* [17], which belongs to the class of null extended data dependencies.

2. THE NULL EXTENDED NESTED RELATIONAL MODEL

In subsection 2.1 we define the data structures of the nested relational model, that is nested relation schemes and nested relations. In subsection 2.2 we present the running example used throughout the paper. In subsection 2.3 we present the operators of the null extended nested relational algebra necessary for our formalism in Section 3 dealing with the nested UR model. In subsection 2.4 we introduce the class of null extended data dependencies, which are integrity constraints that hold in nested relations, and their associated rules. In subsection 2.5 we introduce the *extended chase* procedure; this procedure is used to test satisfaction of a set, D(U), of null extended data dependencies in a nested relation, say r*, over R(T), and to infer more information from r* by using D(U).

2.1. *Nested Relation Schemes and Their Null Extended Nested Relations*

In this subsection we first define scheme trees and their associated nested relation schemes (NRSs). We then define nested relations and an ordering on nested relations, denoted by $\sqsubseteq$, which generalises the *Hoare ordering* on powerdomains [29]. We briefly review the concepts of *greatest lower bound* and *least upper bound* [9] of a set of nested relations over a NRS and show that the set of all nested relations, over a NRS, ordered by $\sqsubseteq$ is a *complete semi-lattice*. Finally, we define the *inequality rule for nulls*, which states that *null* ≠ *null* and then justify our definition.

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 nodes 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 n in T;

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

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

Following Abiteboul and Bidoit [1], we next define the NRS represented by a scheme tree, T.

DEFINITION 2.2. The 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 $R(T) = \Lambda$ (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 $T_1, T_2, ..., T_s$, $s \ge 1$, denote the first level subtrees of the scheme tree, T, with corresponding attributes $(R(T_1))^*$, $(R(T_2))^*$, ..., $(R(T_s))^*$, then $R(T) = X \cup \{(R(T_1))^*, (R(T_2))^*, ..., (R(T_s))^*\}$ (i.e. we denote the attributes of R(T) associated with NRSs $R(T_i)$, $1 \le i \le s$, by $(R(T_i))^*$).

For notational convenience we also represent the NRS R(T) by the string $X(R(T_1))^*(R(T_2))^*...(R(T_s))^*$. The empty string, Λ, is retained in the substring, $(R(T_1))^*(R(T_2))^*...(R(T_s))^*$, 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 $X' \subseteq X$. In the sequel, we use the same notation A to indicate both the single attribute A and the singleton {A}.

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). Λ is neither a zero order nor a higher order attribute.

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 [28, 31].

EXAMPLE 2.1. Let T be the scheme tree over W = {AIRLINE, AIR_CODE, FLIGHT_NO, PASSENGER, CREW, AIRPORT, PORT_CODE}, shown in Fig. 1.1; thus, we have S(T) = W. Moreover, we have the NRS R(T) = AIRLINE AIR_CODE (FLIGHT_NO (PASSENGER)* (CREW)*)* (AIRPORT PORT_CODE)*.

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.

A *nested database scheme* (NDS), **R**(F), over U, is a set {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 set {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 now define null extended nested relations (abbreviated to nested relations). We begin by constructing the underlying powerdomain whose elements are nested relations. For the sake of simplicity, the construction is effected via a single set *Dom* (i.e. the underlying domain of each attribute A_i of the universe U is the set *Dom*). *Dom* is a countable flat domain [9] consisting of atomic values and a generic unmarked null, denoted by *null*, which is taken to be the bottom element of *Dom*; thus *null* contains less information than any other value in *Dom*. In our formalism we consider only the one null value, *null*, and justify this choice by our desire to investigate only the fundamental semantics which are common to all unmarked null types. The induced partial order on *Dom*, denoted as ≤, is defined by by

$\forall\, v_i, v_j \in Dom\;\; v_i \leq v_j$ if and only if $v_i = v_j$ or $v_i = null$ [9].

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)) = {*null*};

(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, 1 ≤ m ≤ p, then DOM(X) = Dom^m, where Dom^m is the Cartesian product (×) of *Dom* with itself m times;

(3) let X = ATT(ROOT(R(T))) and let T_1, T_2, ..., T_s, s ≥ 1, denote the first level subtrees of the scheme tree T. Then, DOM(R(T)) = DOM(X) × P(DOM(R(T_1))) × P(DOM(R(T_2))) × ... × P(DOM(R(T_s))), where P stands for the non-empty finite powerset operator.

We now define a *null extended nested relation*, r* (abbreviated to nested relation), over a NRS R(T), as an element of P(DOM(R(T))), i.e. a (non-empty) finite set of tuples over R(T). 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 $\subseteq$ U, that is R(T) is a FRS. In the sequel, we use the same notation t to indicate both the single tuple t ∈ r* and the singleton {t}.

EXAMPLE 2.2. A nested relation, r*, over R(T) of Example 2.1, is shown in Fig. 1.2.

We observe that by disallowing the empty set in nested relations we have avoided the controversial issue of giving semantics to unnesting the empty set. The semantics of the empty set can best be captured by {*null*}. We note that the empty set is often excluded from powerdomains [29, p. 292].

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*. If r* = {*null*} then we consider r* to be undefined, otherwise we consider r* to be defined.

We define *projection* of a tuple $t \in r^*$ onto $Y \subseteq R(T)$, 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 define the projection of r* onto Y, denoted as r*[Y], by $\{t[Y] \mid t \in r^*\}$.

A *null extended nested database* (abbreviated to nested database), d*, over a NDS, **R**(F), is defined by $d^* = \{r_1^*, r_2^*, ..., r_q^*\}$, where each r^*_i is a defined nested relation over R(T_i), $1 \le i \le q$. 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 $R_i \subseteq U$, $1 \le i \le q$, 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* [29]. We note that the Hoare ordering was also used in [4] to define an ordering over complex objects and the *Smyth* and *Egli-Milner* orderings [29], not considered herein, have been utilised in [7] in the context of generalising flat relational databases.

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

(1) *null* ≤ t and t ≤ t, for any tuple, t, over R(T);

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

(2.1) $\forall A_i \in X$, $t_1[A_i] \le t_2[A_i]$; 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 $w_1 \le w_2$, 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 $t_1 \le t_2$; 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^*.

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 $t_1 \le t_2$ and $t_2 \le t_1$. Correspondingly, a nested relation r_1^* over R(T) is *information-wise equivalent* to a nested relation r_2^* over R(T), denoted by $r_1^* \cong r_2^*$, if and only if $r_1^* \sqsubseteq r_2^*$ and $r_2^* \sqsubseteq r_1^*$.

As a consequence of the above definition of $\cong$ we consider the information content of all tuples and all nested relations in an equivalence class induced by $\sqsubseteq$ to be the same. Thus, $\sqsubseteq$ is a partial order (i.e. a reflexive, antisymmetric and transitive "relation") within the equivalence classes of nested relations with respect to $\cong$. From now on, for simplicity, we refer to $\sqsubseteq$ as a partial order over the set of nested relations.

A minimal element in each of the equivalence classes of nested relations with respect to $\cong$ can be obtained by the process of *reduction*, which we now define. A *reduced nested relation* is a nested relation from which we have removed all tuples which are *less informative* than other distinct (with respect to $\cong$) tuples in the nested relation. 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*. Hereafter, we assume that all nested relations (and thus also flat relations), whether they be given or generated, are *reduced*.

We now briefly review the concepts of *greatest lower bound* (GLB) and *least upper bound* (LUB) [9] of a set of nested relations over a NRS R(T), partially ordered by $\sqsubseteq$. These concepts will be used in Section 3 in the development of the nested UR model.

The GLB (if it exists) of a set of nested relations $\mathbf{S} \subseteq P$(DOM(R(T))) is a nested relation, r*, over R(T), such that:

(1) $\forall$ s* $\in$ **S**, r* $\sqsubseteq$ s*; and

(2) $\forall$ r $\in$ *P*(DOM(R(T))), if $\forall$ s* $\in$ **S**, r $\sqsubseteq$ s*, then r $\sqsubseteq$ r*.

LEMMA 2.1. *The GLB of any non-empty set of nested relations* $\mathbf{S} \subseteq$ *P(DOM(R(T))) exists*.

Proof. We first refer the reader to Definition A1.10 of the null extended meet operator, given in Appendix 1, and note that it can be extended in a straightforward way to a set of nested relations. The result now follows, since it can easily be verified that the null extended meet of all the nested relations in **S**, i.e. $\cap^{ne} \{s_i^* \mid s_i^* \in \mathbf{S}\}$, computes the GLB of **S**. □

We observe that the undefined nested relation, {*null*}, is the GLB of *P*(DOM(R(T))), i.e. it is its bottom element. Furthermore, if $\mathbf{S} = \emptyset$, then the GLB of **S** exists only if *Dom* is finite and in this case the GLB of **S** is equal to DOM(R(T)); if *Dom* is infinite, then DOM(R(T)) is not a nested relation.

The LUB (if it exists) of a set of nested relations $\mathbf{S} \subseteq P$(DOM(R(T))) is a nested relation, r*, over R(T), such that:

(1) $\forall$ s* $\in$ **S**, s* $\sqsubseteq$ r*; and

(2) $\forall$ r $\in$ *P*(DOM(R(T))), if $\forall$ s* $\in$ **S**, s* $\sqsubseteq$ r, then r* $\sqsubseteq$ r.

LEMMA 2.2. *The LUB of any set of nested relations* $\mathbf{S} \subseteq$ *P(DOM(R(T))) exists if and only if* **S** *is a finite set*.

Proof. *IF*. In the case that **S** = ∅, then the LUB of **S** is {*null*}, i.e. the bottom element of *P*(DOM(R(T))). Next we refer the reader to Definition A1.3 of the null extended union operator, given in Appendix 1, and note that it can be extended in a straightforward way to a set of nested relations. The result now follows, since it can easily be verified that the null extended union of all the nested relations in **S**, i.e. $\cup^{ne}$ $\{s_i^* \mid s_i^* \in \mathbf{S}\}$, computes the LUB of **S**.

ONLY IF. Suppose that s* is the LUB of the infinite set **S**, then s* is itself infinite. Thus a contradiction arises, since s* ∈ *P*(DOM(R(T))) must also hold (recall that *P* is the non-empty finite powerset operator), which implies that s* is a nested relation, i.e. a (non-empty) finite set of tuples. □

A partially ordered set, say **D**, is a *complete lattice* if and only if ∀ **S** ⊆ **D**, both the GLB and LUB of **S** exist with respect to the given partial order [9]. If only the GLB exists for **S** ≠ ∅, we say that the partially ordered set is a *complete semi-lattice* (called an *intersection structure* in [9]). The following theorem now follows directly from Lemma 2.1.

THEOREM 2.3. *The set of all nested relations, i.e. P(DOM(R(T))), partially ordered by* $\sqsubseteq$*, is a complete semi-lattice.* □

We note that if we add a top element to *P*(DOM(R(T))), then by Theorem 2.16 in [9] *P*(DOM(R(T)) would become a complete lattice.

We next 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 apply the following rule, referred to as the *inequality rule for nulls*:

(1) if ($v_1 \cong$ *null* and ¬($v_2 \cong$ *null*)) or ($v_2 \cong$ *null* and ¬($v_1 \cong$ *null*)), then $v_1 \neq v_2$;

(2) if $v_1 \cong$ *null* and $v_2 \cong$ *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 [16] due to the inequality rule for nulls. 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 note that, where t_1 and t_2 are tuples in a nested relation, although $t_1[\Lambda] \cong$ *null* and $t_2 \cong$ *null*, we take $t_1[\Lambda] = t_2[\Lambda]$ to be true, since Λ is neither a zero order nor a higher order attribute.

2.2. *The Running Example*

In this subsection we give an example describing details of flight bookings; it will be used throughout the paper. Let U = {PASSENGER, FLIGHT_NO, DATE, DEPENDENT, TICKET_NO, FROM, TO} be the universal set of attributes. The semantics of U are as follows: A PASSENGER is associated with zero or more FLIGHT_NOs and independently zero or more DEPENDENTs and zero or more TICKET_NOs for the booking. In addition, a FLIGHT_NO has one DATE on which the flight departs and several stopovers indicated by pairs FROM city TO city.

Now, let F = $\{T_1, T_2, T_3\}$ be the scheme forest for the running example, where T_1, T_2 and T_3 are shown in Figs. 2.1, 2.2 and 2.3, respectively. The NDS, over U, is **R**(F) = {R(T_1), R(T_2), R(T_3)}.

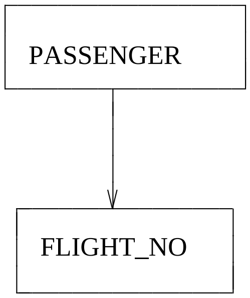

FIG. 2.1. The scheme tree, T_1.

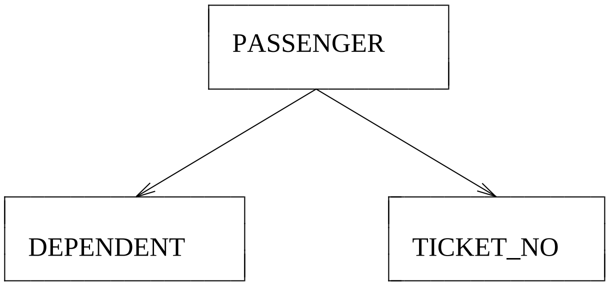

FIG. 2.2. The scheme tree, T_2.

Let d* = $\{r_1^*, r_2^*, r_3^*\}$ be a nested database, over the NDS, **R**(F), where r_1^*, r_2^* and r_3^* are shown in Figs. 2.4, 2.5 and 2.6, respectively.

2.3. *The Null Extended Nested Relational Algebra*

In this subsection we present the operators of the null extended nested relational algebra (or simply the null extended algebra) necessary for our formalism in Section 3 dealing with the nested UR model; the formal definitions of these operators are given in Appendix 1. The main motivation for using the null extended algebra (full details can be found in [19]), as opposed to using one of the existing nested relational algebras from the literature, is that in order to formulate queries with any of the other existing algebras the structure of the nested relations in the nested database needs to be known, whilst the null extended algebra presented herein, frees the user from navigation within the individual nested relations in the nested

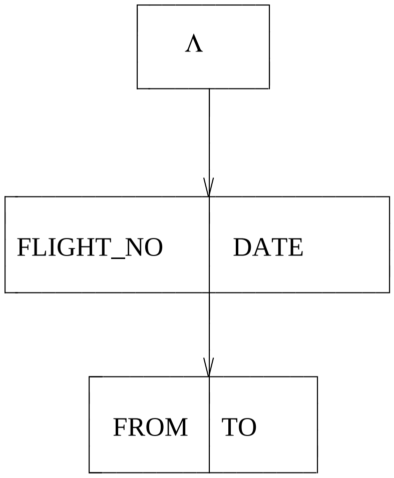

FIG. 2.3. The scheme tree, T_3.

PASSENGER	(FLIGHT_NO)*
	FLIGHT_NO
Iris	213 214
Mark	213
Sara	213
Robert	*null*
Reuven	213 312
Dan	*null*
Mark	214
Iris	312

FIG. 2.4. The nested relation r_1^* over R(T_1).

database. Consequently, using the null extended algebra should not be substantially more difficult than using the standard relational algebra.

We assume the reader is familiar with the operators of the flat relational algebra [8, 33]. In particular, we denote the union operator by $\cup$, the intersection operator by $\cap$, the projection operator by Π, and the natural join operator (or simply the join operator) by $\bowtie$.

An extended algebra is said to be *minimal* if it consists of the flat relational algebra, extended in a natural way to nested relations, i.e., tuples are considered as indivisible units. On the other hand, if some or all of the operators of the flat relational algebra are extended to take advantage of the nested structure of tuples in nested relations then the extended algebra is said to be *maximal*. Although the null extended algebra is a maximal extended algebra as are the algebras defined in [1, 28], we define the null extended union

PASSENGER	(DEPENDENT)*	(TICKET_NO)*
	DEPENDENT	TICKET_NO
Iris	Hanna Brian	111 222
Mark	*null*	333
Mark	*null*	444
Robert	Anette Reuven	*null*
Richard	*null*	*null*
null	Richard	*null*
Robert	*null*	111
Iris	Dan	*null*

FIG. 2.5. The nested relation r_2^* over R(T_2).

Λ	(FLIGHT_NO	DATE	(FROM	TO)*)*
			FROM	TO
null	213	7.3.91	London	Paris
null	213	*null*	Auckland	Los-Angeles
			Los-Angeles	London
null	214	21.8.91	Paris	London
null	312	12.9.91	*null*	Tel-Aviv
			Tel-Aviv	*null*
null	*null*	22.4.92	London	Amsterdam

FIG. 2.6. The nested relation r_3^* over R(T_3).

in a minimal way as is done in the minimal algebras found in [31, 34]. That is to say it corresponds naturally to the standard union operator whereby tuples in a nested relation are considered as indivisible units. The rest of the operators of the null extended algebra are defined in a maximal way, i.e. they are defined recursively to take into account the structure of tuples in a nested relation. That is to say the definitions are the same as those of the standard relational algebra operators when applied to flat relations over zero order attributes; however, these definitions are applied recursively to the higher order attribute values of a nested relation until they reduce to their zero order counterparts. In particular, the following operators are formally defined in Appendix 1:

- The null extended NEST, null extended UNNEST and null extended UNNEST* operators, which are the restructuring operators of the null extended algebra.

- The null extended union operator mentioned above.
- The null extended projection, which extends the projection operator to nested relations, and the null extended total projection, which retains only *total tuples* from the result of the null extended projection, i.e. tuples containing no *nulls*.
- The null extended join operator, which extends the natural join operator to nested relations, wherein tuples are joined at all heights of nodes in the scheme trees of a *joinable NDS* (see Definition 3.1).
- The null extended meet operator, which computes the *meet* [9] of two nested relations (i.e. the GLB of two nested relations).
- The PAD operator, which allows us to pad tuples over a *projected NRS* (see Definition A1.4 in Appendix 1), R(T′), of a NRS, say R(T), with *nulls* so that it be over the NRS, R(T).

2.4. *Null Extended Data Dependencies*

In this subsection we introduce the class of *null extended data dependencies*, which are integrity constraints that hold in nested relations, and their associated rules, which are applied by the extended chase procedure defined in subsection 2.5. These null extended data dependencies are used to develop the nested UR model in Section 3. For more details the reader is referred to [17]. We also mention the counterparts of the class of null extended data dependencies, called hereafter the class of *null data dependencies*, which hold in the flat relations corresponding to the said nested relations.

We first generalise the standard *functional dependency* (FD) [33] that holds in flat relations without *nulls* to the *null FD* (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 define a NFD-rule used for computing the extended chase of a nested relation with respect to a set of null extended data dependencies.

DEFINITION 2.6. (cf. [3, 22].) 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*).

DEFINITION 2.7. 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 $\mu^*(\Pi^{ne}{}_{XA}(r^*))$ prior to its being reduced during the computation of Π^{ne} and μ^*.

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); this set is determined by the semantics of the application under consideration. We observe that Definition 2.7 coincides with the standard definition of the FD in the absence of nulls.

EXAMPLE 2.3. For the scheme tree, T, shown in Fig. 1.1., we have: FF(T) = {AIRLINE $\rightarrow$ AIR_CODE, AIR_CODE $\rightarrow$ AIRLINE, AIRPORT $\rightarrow$ PORT_CODE and PORT_CODE $\rightarrow$ AIRPORT}. It can easily be verified that all the NFDs in FF(T) are satisfied in the nested relation, r*, over R(T), shown in Fig. 1.1. Similarly, for the scheme tree, T_3, shown in Fig. 2.1, we have: FF(T_3) = {FLIGHT_NO $\rightarrow$ DATE}. It can also easily be verified that the NFD FLIGHT_NO $\rightarrow$ DATE is **not** satisfied in the nested relation r_3^*, over R(T_3), shown in Fig. 2.6.

We now define the NFD-rule for a NFD, X $\rightarrow$ A, with respect to a nested relation, r*, over a NRS, R(T).

DEFINITION 2.8. 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 not necessarily distinct XA-tuples in $\mu^*(\Pi^{ne}{}_{XA}(r^*))$ prior to its being reduced during the computation of Π^{ne} and μ^*. 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 $t_1[A] \le t_2[A]$, then $RULE_{X \rightarrow A}(r^*) \cong r'^*$, where r′* is r* after the assignment $t_1[A] := t_2[A]$ has been applied simultaneously to all A-values of the tuples in r* from which t_1 and t_2 originated;

(2) if $t_2[A] \le t_1[A]$, then $RULE_{X \rightarrow A}(r^*) \cong r'^*$, where r′* is r* after the assignment $t_2[A] := t_1[A]$ has been applied simultaneously to all A-values of the tuples in r* from which t_1 and t_2 originated;

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

We now 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 2.7 , 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 2.9. 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 $\exists\, t_1, t_2 \in$ r* such that t_1[X] = t_2[X], i.e. t_1, t_2 are X-total, t_1[(Y)*] $\cong t_2$[(Y)*]; otherwise

(2) let $T_1, T_2, \ldots, T_s$, s $\geq$ 1, denote the first level subtrees of the scheme tree, T. Then, $\forall$ 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).

We note that by the above definition only NEFDs of the form X $\rightarrow$ (R(T_v))* can be satisfied in r*, where (u,v) is an edge in the scheme tree T, X $\subseteq$ A(u) and T_v is the subtree rooted at the node, v, of T.

EXAMPLE 2.4. For the scheme tree, T, shown in Fig. 1.1 we have the set of NEFDs: {AIRLINE,AIRCODE $\rightarrow$ (FLIGHT_NO (PASSENGER)* (CREW)*)*, AIRLINE,AIRCODE,FLIGHT_NO $\rightarrow$ (PASSENGER)*, AIRLINE,AIRCODE,FLIGHT_NO $\rightarrow$ (CREW)*, AIRLINE,AIRCODE $\rightarrow$ (AIRPORT, PORT_CODE)*}. It can easily be verified that this set of NEFDs is satisfied in the nested relation, r*, over R(T), shown in Fig. 1.2. It can also easily be verified that the NEFD, PASSENGER $\rightarrow$ (FLIGHT_NO)*, is **not** satisfied in the nested relation r_1^*, over R(T_1), which is shown in Fig. 2.4.

We now define the NEFD-rule for a NEFD, X $\rightarrow$ (Y)*, with respect to a nested relation, r*, over a NRS, R(T).

DEFINITION 2.10. 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 $\subseteq$ Z(R(T)) and (Y)* $\in$ H(R(T)), and $\exists\, t_1, t_2 \in$ r* such that t_1[X] = t_2[X], i.e. t_1 and t_2 are X-total, and $\neg$(t_1[(Y)*] $\cong t_2$[(Y)*]), then $RULE_{X \rightarrow (Y)*}$(r*) $\cong$ 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 $T_1, T_2, \ldots, T_s$, 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, $r'^*_i \cong$ {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 $RULE_{X \rightarrow (Y)*}(r'^*_i)$.

In Appendix 2 there can be found the formal definition of another member of the class of null extended data dependencies, called the *null extended join dependency* (NEJD), which enables us to capture the notion of lossless decomposition in nested relations. The NEJD is an extension of a member of the class of null data dependencies, called the *null join dependency* (NJD) [21], to nested relations. (NJDS are *join dependencies* (JDs) [5] that hold in flat relations which may contain *nulls*.) We note that the *null multivalued dependency* (NMVD) is a special case of the NJD, i.e. when the decomposition is just two.

We close this subsection with the definition of the set of NMVDs represented in a scheme tree, T, denoted by MVD(T), and the corresponding set of NEFDs represented by a scheme tree T, which is denoted by FD(T). We use the standard notation $X \rightarrow\rightarrow Y$ [33] to refer to a NMVD from $X \subseteq U$ to $Y \subseteq U$. Also, we write $X \rightarrow\rightarrow Y$ (W) to mean $X \rightarrow\rightarrow Y$ in the context of $W \subseteq U$, and, in general, we omit W whenever the context is understood.

DEFINITION 2.11. Let e = (u,v) be an edge in a scheme tree, T, and let T_v be the subtree rooted at the node, v, of T. Then the NMVD represented by the edge, e, is given by MVD(e) = A(u) $\rightarrow\rightarrow$ D(v) (S(T)), and the corresponding NEFD, represented by the edge e, is FD(e) = A(u) $\rightarrow$ (R(T_v))*.

The set of NMVDs, which are represented by all the edges of a scheme tree, T, is denoted by MVD(T), and correspondingly the set of NEFDs, which are represented by all the edges of a scheme tree, T, is denoted by FD(T).

We observe that FD(T) is induced by the structure of the scheme tree, T. Therefore, from a design point of view FD(T) should always be considered as a possible part of the semantics of the application under consideration.

2.5. *The Extended Chase Procedure*

In this subsection we extend the chase procedure [23] to nested relations, and call it the *extended chase* procedure; this procedure is used to test satisfaction of a set of null extended data dependencies, denoted hereafter by D(U), in a nested relation, say r*, over U(T), and to infer more information from r* by using D(U). Henceforth, D will denote the set of *null data dependencies* emanating from D(U). In the context of this paper the set D(U) includes:

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

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

(3) the NEJD, $\bowtie^{ne}[\mathbf{R}(F)]$, over the *joined NRS* (see Definition 3.2), U(T), which we denote by JD(F).

We denote the result of applying the extended chase to a nested relation, r*, over a joined NRS, U(T), with respect to the set D(U), 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 (see subsection 2.4) in D(U) until no more rules can be applied. In the case when $CHASE_{D(U)}(r^*) \cong \{null\}$, i.e. a NFD in D(U) is violated, we say that r* is *inconsistent* with respect to D(U) (or simply inconsistent, when D(U) is understood from context), otherwise we say that r* is *consistent* with respect to D(U) (or simply consistent, when D(U) is understood from context) (cf. [3, 13, 15, 26]). If r* is consistent, then $CHASE_{D(U)}$(r*) satisfies D(U) and is an *inflationary fixpoint* [14] of D(U) on r*; this fixpoint is in fact the *least fixpoint* of D(U) on r*, by Theorem 3 in [14], since the extended chase procedure is monotone for consistent nested relations. Full details can be found in [17].

The next theorem shows that an extended chase of a nested relation, r*, with respect to D(U) is information-wise equivalent to an extended chase of μ*(r*) with respect to D.

THEOREM 2.4. [17]. *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), JD(F)} and D = {FF(T), MVD(T),* $\bowtie$*[FDS(F)]}. Then*

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

Our next corollary, which is a direct consequence of Theorem 2.4, 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 2.5. [17]. *Let r*, D(U) and D be as in Theorem 2.4. Then r* is consistent with respect to D(U) if and only if μ*(r*) is consistent with respect to D.* □

3. THE NESTED WEAK INSTANCE APPROACH TO THE NESTED UR MODEL

We are now ready to formalise the nested UR model and present our main results culminating in Theorem 3.8 wherein it is shown that the weak instance approach is a special case of the nested weak instance approach. In subsection 3.1 we define the concept of a *joinable* NDS and introduce the *nested universal relation scheme* (NURS) of a joinable NDS. In subsection 3.2 we introduce the nested weak instance approach and give a declarative definition of the *nested representative instance* (NRI), which provides the underlying data structure of the nested UR model wherein the semantics of the nested database are encapsulated within a single nested relation. We show that the NRI exists exactly when the set of nested weak instances under a set of null extended data dependencies for a given nested database is non-empty. In subsection 3.3 we give a constructive definition of the NRI via the extended chase procedure and in Theorem 3.6 show its equivalence to the declarative definition of the NRI. Finally, in subsection 3.4 we show the equivalence of the NRI under a set of null extended data dependencies for a given nested database to the RI under the corresponding set of null data dependencies for the corresponding flat database.

3.1. *The Nested Universal Relation Scheme*

The NURS, denoted by U(T), provides the necessary NRS over which null extended joins between nested relations in a nested database are well defined. This is essential for query processing in the nested UR model, since it provides automatic logical navigation amongst the nested relations in the nested database.

In order to formalise the notion of the NURS 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, U(T), over U, which we call the *joined NRS* of **R**(F), without violating the definition of a scheme tree.

DEFINITION 3.1. 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* [1], 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 3.2. 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 ≤ i ≤ q.

EXAMPLE 3.1. Let F be the scheme forest of the running example. It can easily be verified that **R**(F) is a joinable NDS and that U(T), where T is shown in Fig. 3.1, is the joined NRS of **R**(F).

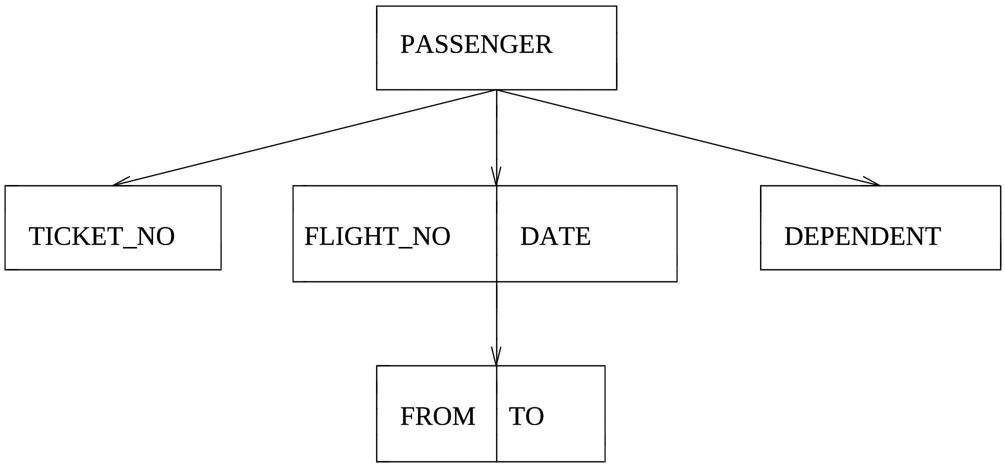

FIG. 3.1. The scheme tree, T, of the joinable NDS, **R**(F).

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 3.1. [19]. *A NDS,* **R***(F), is joinable if and only if there exists a joined NRS, U(T), of* **R***(F).*

Herein, we do not deal with the problem of restructuring an arbitrary NDS (which may not be joinable) into a joinable NDS. A transformation algorithm of a non-joinable NDS into a joinable NDS, which solves this problem, and its correctness are given in [20]. From now on we will assume that all NDSs that we consider are joinable. In particular, the NURS, U(T), will be taken to be the joined NRS of the joinable NDS, **R**(F).

3.2. *The Nested Representative Instance*

In this subsection we present the *nested representative instance* (NRI) [20], which is the single nested relation, over the NURS. The NRI provides the underlying data structure of the nested UR model, wherein the semantics of the nested database are encapsulated in a single nested relation. Thus, the NRI, over the NURS, frees the user from logical navigation amongst and within the nested relations in the nested database, since the user can query the nested database via the NRI, solely through the universal set of attributes.

As before we let U(T) be the NURS of the NDS, **R**(F), where F = $\{T_1, T_2, ..., T_q\}$; we also consider the associated nested database, d* = $\{r_1^*, r_2^*, ..., r_q^*\}$, over the NDS **R**(F), and a set of null extended data dependencies, D(U). The set D(U) includes:

(1) FF(T) which is the set of NFDs, $\cup_{i=1}^{q}$ FF(T_i);

(2) FD(T) which is the set of NEFDs induced by the structure of the scheme tree, T; and

(3) JD(F) which is the NEJD that ensures that the NRI, I*, under D(U) for d*, possesses a *null extended lossless decomposition* onto **R**(F) [17].

In order to formalise the nested UR model, we extend the weak instance approach [3, 13, 15, 24, 26, 27, 30] to nested relations. Informally, a nested weak instance, say I*, is a nested relation over the NURS, U(T), which satisfies D(U), and such that the null extended projections of I* onto the attributes associated with each nested relation, $r_i^* \in$ d*, are more informative than r_i^*. This approach allows us to formalise the nested UR model whilst allowing the associated nested database to be an incomplete description of the real world.

DEFINITION 3.3. A nested relation, I*, over U(T), is a *nested weak instance* under D(U) for d*, if

(1) I* satisfies D(U); and

(2) $r_i^* \sqsubseteq \Pi^{ne}{}_{S(T_i)}(\text{I*})$, $1 \le i \le q$.

As already stated, we denote the set of all nested weak instances, ordered by $\sqsubseteq$, under D(U) for d* by NWI(D(U), d*). We note that if there does not exist a nested relation, I*, that satisfies conditions (1) and (2) of Definition 3.3, then NWI(D(U), d*) = ∅. In the nested weak instance approach to the nested UR model the nested database, d*, is, in general, a partial description of a nested weak instance, I*, over U(T), which satisfies the semantics of d* given in the form of the set D(U). We note that by Definition 3.3 of a nested weak instance we allow null values to appear in any nested weak instance, while it is a common simplifying assumption in the weak instance approach to the classical UR model that the underlying database does not contain null values as is the case in [15].

In the nested weak instance approach, as in the weak instance approach, there may be, in general, countably infinite many nested weak instances under D(U) for d* (or a very large number if the flat domain *Dom* is finite). Thus, we assume that the only information that can be deduced from the said set of nested weak instances, NWI(D(U), d*), under the nested UR model, is the information that holds in *all* nested weak instances under D(U) for d*.

DEFINITION 3.4. A nested relation, I*, over U(T), is the *nested representative instance* (NRI) under D(U) for d*, if I* is the GLB of NWI(D(U), d*).

Hereafter, in the special case when d* is a flat database, say d, and thus D(U) is a set of null data dependencies, D, we refer to the NRI, over U(T), under D(U) for d*, as the *representative instance* (RI), over U, under D for d.

The next theorem shows that the set of nested weak instances ordered by $\sqsubseteq$ is a complete semi-lattice (see the definition of a complete semi-lattice prior to Theorem 2.3).

THEOREM 3.2. *For all non-empty subsets,* **S** $\subseteq$ *NWI(D(U), d*), the GLB of* **S***, say I*, exists and is such that I** $\in$ *NWI(D(U), d*).*

Proof. By Lemma 2.1 it follows that I* exists and is a nested relation. It remains to show that I* $\in$ NWI(D(U), d*). We prove the result by induction on the number, m, of nested weak instances in **S**.

BASIS. If m = 1 then the result holds, since the GLB of **S** is the single nested weak instance, I* $\in$ **S**.

INDUCTION. Assume the result holds for m = n. Then we prove the result for m = n+1. Let **S** = $\{I_1, I_2, ..., I_n, I_{n+1}\}$. Then, by inductive hypothesis, the GLB of $\{I_1, I_2, ..., I_n\}$, say I, is in NWI(D(U), d*). It remains to show that the GLB of I and I_{n+1}, say J, is in NWI(D(U), d*), i.e. we need to show that

(1) J satisfies D(U); and

(2) $r_i^* \sqsubseteq \Pi^{ne}{}_{S(T_i)}(J)$, $1 \leq i \leq q$.

Let D(U) = {FF(T), FD(T), JD(F)}. We prove part (1) by contradiction:

(i) If J does not satisfy FF(T), then there exists a NFD, X $\rightarrow$ A $\in$ FF(T), that is violated in J, i.e. $\exists\, t_1, t_2 \in \mu^*(\Pi^{ne}{}_{XA}(J))$ prior to its being reduced during the computation of Π^{ne} and μ^*, such that t_1[X] = t_2[X] and $\neg(t_1[A] \cong t_2[A])$. This contradicts the fact that I and I_{n+1} satisfy FF(T), since, due to the fact that J is the GLB of I and I_{n+1}, the said violation would imply that X $\rightarrow$ A is also violated in one or both of I and I_{n+1}.

(ii) If J does not satisfy FD(T), then there exists a NEFD, X $\rightarrow$ (Y)* $\in$ FD(T), that is violated in J. 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 J, corresponding to recursive applications of the NEFD-rule, i.e. Definition 2.9(2), thus obtaining, say J′ over R(T′), such that X = Z(R(T′)) and (Y)* $\in$ H(R(T′)).) Thus, $\exists\, t_1, t_2 \in$ J, such that t_1[X] = t_2[X] and $\neg(t_1[(Y)^*] \cong t_2[(Y)^*])$. This contradicts the fact that I and I_{n+1} satisfy FD(T), since, due to the fact that J is the GLB of I and I_{n+1}, the said violation would imply that X $\rightarrow$ (Y)* is also violated in one or both of I and I_{n+1}.

(iii) If J does not satisfy JD(F), then $\exists\, t_1, t_2, ..., t_n \in$ J, which are combinable with the resulting tuple t*, over U(T), but t* $\notin$ J. This contradicts the fact that I and I_{n+1} satisfy JD(F), since, due to the fact that J is the GLB of I and I_{n+1}, t* $\notin$ J would imply that JD(F) is also violated in one or both of I and I_{n+1}.

Part (2) follows since, by inductive hypothesis, $r_i^* \sqsubseteq \Pi^{ne}{}_{S(T_i)}(I)$; also $r_i^* \sqsubseteq \Pi^{ne}{}_{S(T_i)}(I_{n+1})$, $1 \leq i \leq q$, and J is defined to be the GLB of I and I_{n+1}. □

The following corollary, whose result follows immediately from Theorem 3.2 and Definition 3.4, asserts the fact that the NRI is indeed a nested weak instance. Thus, the NRI is the bottom element of the complete semi-lattice induced by the set of nested weak instances NWI(D(U), d*) ordered by $\sqsubseteq$.

COROLLARY 3.3. *If I* is the NRI under D(U) for d*, then I** $\in$ *NWI(D(U), d*).* □

The next corollary, whose result follows from the previous corollary, generalises Lemma 1 of [2] to nested relations.

COROLLARY 3.4. *Let d_1^* and d_2^* be two nested databases over* **R**(*F*) *and let I_1^* and I_2^* be the NRIs, over D(U), under D(U) for d_1^* and d_2^*, respectively. Then, $I_1^* \sqsubseteq I_2^*$ if and only if NWI(D(U), d_2^*) $\subseteq$ NWI(D(U), d_1^*).* □

We note that Atzeni and Torlone [2] define a complete lattice on flat databases in order to formalise the problem of updating flat databases (without nulls) under the weak instance approach. In our case we are interested in the problem of extending the weak instance approach to nested relations in the presence of nulls in the nested database. Thus, the formalisation in [2] is different from ours and does not utilise powerdomains. This different view point enables us later in this section to give an equivalent constructive definition of the NRI via the extended chase procedure.

We observe that if we add a distinguished top element to the set of nested weak instances representing an overdetermined nested relation, then NWI(D(U), d*) ordered by $\sqsubseteq$ would be promoted from a complete semi-lattice to a complete lattice (see the remark after Theorem 2.3).

We now relate the result obtained in Corollary 3.3 to a similar result obtained in [18] for the weak instance approach. Therein, the representative instance is shown to be the GLB of the set of weak instances under a set of FDs for a flat database.

Let $X \subseteq U$; the window, [X], for the NRI, I*, under D(U) for d*, is defined by

$$[X] \cong \mu^*(\Pi^{ne}\downarrow_X(I^*)).$$

That is, under the nested weak instance approach, the *window*, [X], *for a set of attributes X ⊆ U, contains exactly the X-total tuples that appear in every nested weak instance I* under D(U) for d*.* The following proposition formalises this fact.

PROPOSITION 3.5. *Suppose that NWI(D(U), d*) ≠ ∅, then for any set of attributes, X ⊆ U,*

$$[X] = \cap_{I^* \in NWI(D(U), d^*)} \mu^*(\Pi^{ne}\downarrow_X(I^*)).$$

Proof. By Corollary 3.3, the NRI I* under D(U) for d* is in NWI(D(U), d*). The result now follows directly from Definition 3.4 of the NRI, since the NRI is the GLB of NWI(D(U), d*). □

We observe that the above proposition generalises the same result obtained for the weak instance approach to the classical UR model in Theorem 1 of [24]. Due to the fact that we define the NRI to be the GLB of all nested weak instances, our proof is simpler than the one given in [24].

It has been argued by Ullman [32] that null values should also be included in the window, [X], as information is lost, at times, when only total tuples are considered and output. It is, therefore, possible to redefine [X] by

$$[X] \cong \mu^*(\Pi^{ne}{}_X(I^*)),$$

in order to include null values. In this case it follows that

$$[X] \cong \cap^{ne}{}_{I^* \in NWI(D(U), d^*)} \mu^*(\Pi^{ne}{}_X(I^*)),$$

i.e. the null extended meet operator computes the window for X over the set of nested weak instances rather than the intersection operator as in Proposition 3.5.

3.3. *The Constructive NRI*

In this subsection we give a constructive definition of the NRI, which we refer to as the *Constructive NRI* (CNRI). We then present one of the main results of the paper, i.e. that the constructive definition of the NRI (Definition 3.5) is equivalent to the declarative definition of the NRI (Definition 3.4). Thus, the extended chase provides us with an effective tool to construct the NRI under D(U) for d*.

DEFINITION 3.5. We define the CNRI, I*, over U(T), under D(U) for d*, by

$$I^* \cong CHASE_{D(U)}(\mathrm{PAD}(d^*)).$$

If I* ≅ {*null*}, then I* is said to be *inconsistent*, otherwise it is said to be *consistent*. The following theorem proves the equivalence of the NRI and the CNRI.

THEOREM 3.6. *Let J*, over U(T), be the NRI under D(U) for d* on using Definition 3.4 and let I*, over U(T), be the CNRI under D(U) for d* on using Definition 3.5. Then, assuming that I* is consistent, we have that J* ≅ I*.*

Proof. In order to prove the result that J* ≅ I* we need to show that the CNRI, I*, is also the NRI, i.e. that I* is the GLB of NWI(D(U), d*) ordered by $\sqsubseteq$ or, equivalently, that ∀ I ∈ NWI(D(U), d*), I* $\sqsubseteq$ I.

We first observe that I* ∈ NWI(D(U), d*), since I* satisfies Definition 3.3 of a nested weak instance on using the definition of the extended chase procedure. Now, suppose that I* is not the NRI, i.e. that ¬(J* ≅ I*) and thus ∃ J ∈ NWI(D(U), d*) such that J $\sqsubseteq$ I* and ¬(I* $\sqsubseteq$ J). It follows that ∃ t ∈ I* such that ∄ w ∈ J satisfying t ≤ w. Since J must satisfy Definition 3.3(1) of a nested weak instance under D(U) for d*, it follows that t ∉ $CHASE_{D(U)}$(PAD(d*)) because the extended chase procedure yields the least fixpoint of D(U) on PAD(d*). This contradicts the fact that I* is the CNRI, thus proving the result. □

We remark that as a direct consequence of Theorem 3.6 the window defined on the NRI satisfies the *containment condition* that Π_X([Y]) ⊆ [X] holds, whenever X ⊆ Y [25].

DEFINITION 3.6. A nested database d* is *inconsistent* with respect to D(U) (or simply inconsistent, when D(U) is understood from context), if and only if NWI(D(U), d*) = ∅; otherwise, we say that d* is *consistent* with respect to D(U) (or simply consistent, when D(U) is understood from context).

We can now characterise the consistency of the CNRI according to the consistency of d*. This result can be viewed as extending Theorem 1 of [15] to nested relations.

THEOREM 3.7. *d* is inconsistent if and only if the CNRI, say I* over U(T), under D(U) for d* is inconsistent.*

Proof. *IF*. If the CNRI, I*, is inconsistent, then I* ≅ {*null*}. It therefore follows that Definition 3.3 cannot be satisfied and thus NWI(D(U), d*) = ∅ as required.

ONLY IF. If d* is inconsistent, then NWI(D(U), d*) = ∅. It therefore follows from Theorem 3.6 that I* is inconsistent, otherwise we would have I* ∈ NWI(D(U), d*) by Corollary 3.3. □

We observe that the above result implies that d* is consistent if and only if the NRI under D(U) for d* exists. We next give an example illustrating the construction of the NRI.

EXAMPLE 3.2. Let d* be the nested database, over the NDS, **R**(F), of the running example, and let U(T), where T is shown in Fig. 3.1, be the NURS of **R**(F).

Let FF(T) = {FLIGHT_NO $\rightarrow$ DATE}, let FD(T) = {PASSENGER $\rightarrow$ (TICKET_NO)*, PASSENGER $\rightarrow$ (DEPENDENT)*, PASSENGER $\rightarrow$ (FLIGHT_NO,DATE(FROM,TO)*)*, PASSENGER,FLIGHT_NO,DATE $\rightarrow$ (FROM,TO)*}, and let JD(F) = $\bowtie^{ne}$[{PASSENGER(FLIGHT_NO)*, PASSENGER(DEPENDENT)*(TICKET_NO)*, Λ(FLIGHT_NO,DATE(FROM,TO)*)*}]. Thus D(U) = {FF(T), FD(T), JD(F)].

The nested relation PAD(d*) is shown in Fig. 3.2 and the NRI, I* $\cong$ $CHASE_{D(U)}$(PAD(d*)) , over U(T), under D(U) for d* is shown in Fig. 3.3.

3.4. *Equivalence of the NRI and the RI*

In this subsection we show the equivalence of the NRI for a nested database with the RI for a flat database. More specifically, given a nested database, d*, over a NDS, **R**(F), and the set D(U), we prove in Theorem 3.8 the following result. Let d be the flat database obtained by applying the UNNEST* operator to all the nested relations in d* and let D be the set of null data dependencies emanating from D(U). Then, the NRI, over the NURS, U(T), under D(U) for d*, is equivalent to the RI over the universal set of attributes, U, under D for d. The implication of this result is that the classical UR model under the weak instance approach is a special case of the nested UR model under the nested weak instance approach.

THEOREM 3.8. *Let* **R**(F) *be a joinable NDS and let D(U) = {FF(T), FD(T), JD(F)}. Then, the following two statements are equivalent:*

(1) *I*, over U(T), is the NRI under D(U) for d*.*

(2) μ*(I*), over U, is the RI under the set D = {FF(T), MVD(T), $\bowtie$[FDS(F)]}, for the flat database d = {$\mu^*(r_1^*)$, $\mu^*(r_2^*)$, ..., $\mu^*(r_q^*)$}, over the FDS, FDS(F).*

Proof. We first observe that d* is consistent with respect to D(U) if and only if d is consistent with respect to D by Corollary 2.5. Hence, if d* and d are inconsistent then the NRI under D(U) for d* and the RI under D for d are both inconsistent. Thus, we assume that d* and d are consistent.

By Definition 3.5 we have that

$$I^* \cong CHASE_{D(U)}(PAD\,(d^*)),$$

so, by Theorem 2.4, we obtain

PASSENGER	(TICKET_NO)*	(FLIGHT_NO	DATE	(FROM	TO)*)*	(DEPENDENT)*
	TICKET_NO	FLIGHT_NO	DATE	(FROM	TO)*	DEPENDENT
				FROM	TO	
Iris	*null*	213	*null*	*null*	*null*	*null*
		214	*null*	*null*	*null*	
Iris	*null*	312	*null*	*null*	*null*	*null*
Iris	111 222	*null*	*null*	*null*	*null*	Hanna Brian
Iris	*null*	*null*	*null*	*null*	*null*	Dan
Mark	*null*	214	*null*	*null*	*null*	*null*
Mark	*null*	213	*null*	*null*	*null*	*null*
Mark	333	*null*	*null*	*null*	*null*	*null*
Mark	444	*null*	*null*	*null*	*null*	*null*
Sara	*null*	213	*null*	*null*	*null*	*null*
Reuven	*null*	213	*null*	*null*	*null*	*null*
	null	312	*null*	*null*	*null*	
Dan	*null*	*null*	*null*	*null*	*null*	*null*
Robert	*null*	*null*	*null*	*null*	*null*	Anette Reuven
Robert	111	*null*	*null*	*null*	*null*	*null*
Richard	*null*	*null*	*null*	*null*	*null*	*null*
null	*null*	*null*	*null*	*null*	*null*	Richard
null	*null*	213	7.3.91	London	Paris	*null*
null	*null*	213	*null*	Auckland	Los-Angeles	*null*
				Los-Angeles	London	
null	*null*	214	21.8.91	Paris	London	*null*
null	*null*	312	12.9.91	*null*	Tel-Aviv	*null*
				Tel-Aviv	*null*	
null	*null*	*null*	22.4.92	London	Amsterdam	*null*

FIG. 3.2. The nested relation PAD(d*).

$$\mu^*(I^*) \cong CHASE_D(\mu^*(PAD(d^*))).$$

It can easily be verified that

$$\mu^*(PAD(d^*)) \cong PAD(d).$$

PASSENGER	(TICKET_NO)*	(FLIGHT_NO	DATE	(FROM	TO)*)*	(DEPENDENT)*
	TICKET_NO	FLIGHT_NO	DATE	(FROM	TO)*	DEPENDENT
				FROM	TO	
Iris	111	213	7.3.91	London	Paris	Hanna
	222			Auckland	Los-Angeles	Brian
				Los-Angeles	London	Dan
		214	21.8.91	Paris	London	
		312	12.9.91	*null*	Tel-Aviv	
				Tel-Aviv	*null*	
Mark	333	214	21.8.91	Paris	London	*null*
	444	213	7.3.91	London	Paris	
				Auckland	Los-Angeles	
				Los-Angeles	London	
Sara	*null*	213	7.3.91	London	Paris	*null*
				Auckland	Los-Angeles	
				Los-Angeles	London	
Reuven	*null*	213	7.3.91	London	Paris	*null*
				Auckland	Los-Angeles	
				Los-Angeles	London	
		312	12.9.91	*null*	Tel-Aviv	
				Tel-Aviv	*null*	
Dan	*null*	*null*	*null*	*null*	*null*	*null*
Robert	111	*null*	*null*	*null*	*null*	Anette Reuven
Richard	*null*	*null*	*null*	*null*	*null*	*null*
null	*null*	*null*	*null*	*null*	*null*	Richard
null	*null*	*null*	22.4.92	London	Amsterdam	*null*

FIG. 3.3. The NRI, I*, over U(T), under D(U) for d*.

Thus, we now have that

$$\mu^*(I^*) \cong CHASE_D(PAD(d)). \quad (1)$$

This concludes the result since, by applying Theorem 3.6 to (1), it follows that μ*(I*) is the RI under D for d and that I* is the NRI under D(U) for d*. □

EXAMPLE 3.3. For the NDS, **R**(F), of the running example and U(T) of Example 3.1, let FDS(F) = {S(T_1), S(T_2), S(T_3)} be a FDS over U = {PASSENGER, TICKET_NO, FLIGHT_NO, DATE, FROM, TO, DEPENDENT} with d = {μ*(r_1^*) μ*(r_2^*) μ*(r_3^*)} being the corresponding flat database over U. Also, let FF(T) = {FLIGHT_NO $\rightarrow$ DATE}, MVD(T) = {PASSENGER $\rightarrow\rightarrow$ TICKET_NO (S(T)), PASSENGER $\rightarrow\rightarrow$ DEPENDENT (S(T)), PASSENGER,FLIGHT_NO,DATE $\rightarrow\rightarrow$ FROM,TO (S(T))} and $\bowtie$[FDS(F)] = $\bowtie$[{{PASSENGER,FLIGHT_NO}, {PASSENGER,DEPENDENT,TICKET_NO}, {FLIGHT_NO,DATE,FROM,TO}}]. Thus D = {FF(T), MVD(T), $\bowtie$[FDS(F)]}. It can easily be verified that μ*(I*), over U, is the RI under D for d, where I*, over U(T), shown in Fig. 3.2, is the NRI under D(U) for d*, of Example 3.2.

The essential consequence of Theorem 3.8 is that it guarantees that a UR interface can be implemented by using the nested UR model, thus gaining all the advantages of nested relations over flat relations. This UR interface can provide both flat and hierarchical output to the user at the external level. In addition, since the nested UR model is more expressive than the classical UR model, the range of applications that can be naturally modelled within the nested UR model is much larger than the corresponding range of applications that can be modelled within the classical UR model.

4. CONCLUSIONS

We have presented the nested UR model in order to alleviate the usability problem for nested relations and thus to provide the nested relational model with logical data independence. In particular, we extended the weak instance approach to the classical UR model to the nested weak instance approach to the nested UR model. We first showed (in Theorem 3.2) that NWI(D(U), d*) ordered by $\sqsubseteq$ is a complete semi-lattice. Thus, by Corollary 3.3 the GLB of NWI(D(U), d*) exists and is in NWI(D(U), d*) provided this set is non-empty. This GLB, which is the bottom element of the complete semi-lattice induced by NWI(D(U), d*), when ordered by $\sqsubseteq$, provides us with the declarative definition of the NRI under D(U) for d* (Definition 3.4). We then gave a constructive definition of the NRI via $CHASE_{D(U)}$(PAD(d*)) (Definition 3.5). In Theorem 3.6 we showed that the declarative and constructive definitions of the NRI are equivalent. Thus, the extended chase gives the NRI its operational semantics while the GLB of the set NWI(D(U), d*) ordered by $\sqsubseteq$ gives the NRI its denotational semantics. Furthermore, in Theorem 3.7 we showed that a nested database is consistent with respect to D(U) if and only if the CNRI under D(U) for d* is consistent. Our final result, given in Theorem 3.8, shows that the weak instance approach is a special case of the nested weak instance approach. This result is important, since it allows us to gain all the advantages of nested relations over flat relations in the implementation of a UR interface under the nested UR model. Thus, we have shown that the nested weak instance approach is *upwards compatible* with the weak instance approach.

In constructing the NRI via the extended chase, we face the intractability, in general, of the computational complexity of the extended chase (cf. [13]); thus from a practical point of view it may not be feasible to construct the NRI in this way. In [20] we have investigated an algebraic computational method which constructs the NRI by employing the null extended algebra, whenever FDS(F) is γ-*acyclic* [11]. Thus, in this special case, a DBMS supporting the null extended algebra, but not necessarily supporting the extended chase, can effectively support the nested UR model. By using unmarked nulls, it was shown in [3] that the weak instance approach allows all of the known computational approaches to the RI, given in the literature, such as the *independent database schemes* of [12, 27], to be supported. Thus, as a consequence of Theorem 3.8 the nested weak instance approach can also support these computational approaches.

APPENDIX 1. THE NULL EXTENDED NESTED RELATIONAL ALGEBRA

Herein, we give the formal definitions of the following operators of the null extended nested relational algebra: NEST, UNNEST and UNNEST*, null extended union, null extended projection and total projection, null extended join, null extended meet and PAD.

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

DEFINITION A1.1. Let r* be a nested relation over R(T) and let Y ($\neq \Lambda$) $\subseteq$ R(T). ν_Y(r*) is a nested relation over (R(T) − Y)(Y)* such that a tuple w $\in$ ν_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)*] $\cong$ {t′[Y] | t′ $\in$ r* and t′[R(T) − Y] $\cong$ w[R(T) − Y]}.

DEFINITION A1.2. 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 [19] (cf. [31]), denoted by μ*, transforms any nested relation, r*, into a flat relation. Thomas and Fischer [31] showed that the order of unnesting does not affect the resulting flat relation, μ*(r*).

From now on, for the sake of simplicity, we shall simply call the null extended NEST, NEST, the null extended UNNEST, UNNEST and the null extended UNNEST*, UNNEST*.

DEFINITION A1.3. The *null extended union*, $\cup^{ne}$, of two nested relations, r_1 and r_2, over R(T), is defined by:

$r_1 \cup^{ne} r_2 \cong \{t \mid t \in r_1 \vee t \in r_2\}$.

We next 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.

DEFINITION A1.4. 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.

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 [6], since no restrictions are placed on the set of attributes of S(T) over which the projection takes place.

A useful syntax for projected NRSs now follows.

DEFINITION A1.5. 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.

We are now ready to define the *null extended projection* operator, Π^{ne}. Given a nested relation r* over a NRS, R(T), Π^{ne}_Y(r*), where Y ⊆ 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 ∈ r*.

DEFINITION A1.6. Let R(T′) be a projected NRS of R(T) over Y ⊆ S(T), and let r* be a nested relation over R(T). Then the *null extended projection* of r* onto Y, Π^{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 Π^{ne}_Y(r*) ≅ {t[Y] | t ∈ r*}, where Y ⊆ X;

(2) if R(T) = X(R(T_1))*(R(T_2))* ... (R(T_s))*, where X = ATT(ROOT(T)) and T_1, T_2, ..., T_s, s ≥ 1, denote the first level subtrees of T, let R(T′) = X′(R(T'_1))*(R(T'_2))* ... (R(T'_w))*, where X′ = ATT(ROOT(T′)) and

w ≤ s, be a projected NRS of R(T). Then,

$\Pi^{ne}{}_{Y}(r^*) \cong \{t \mid \exists\ t' \in r^* \wedge t[X'] \cong t'[X'] \wedge$

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

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

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 ⊆ S(T), by

$t[Y] \cong \Pi^{ne}{}_{Y}(t)$.

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

The next definition deals with the concept of X-total tuples in a nested relation, for a set of attributes, X ⊆ S(T). Let r* be a nested relation over R(T) and t ∈ 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 ⊆ S(T), then we say that the tuple t is *X-total* if and only if X ⊆ DEF(t).

DEFINITION A1.7. Let r* be a nested relation over a NRS, R(T). Then, for t ∈ r*, DEF(t) is defined recursively by:

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

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

(2) 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, 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 [27, 30].

We next define the *null extended total projection*, denoted as $\Pi^{ne}\downarrow$. Given a nested relation r* over a NRS, R(T), $\Pi^{ne}\downarrow_Y(r^*)$, where Y ⊆ S(T), intuitively returns the largest subset of r*, over a projected NRS, of R(T), R(T′), with S(T′) = Y, such that it contains only Y-total tuples.

DEFINITION A1.8. Let R(T′) be a projected NRS of R(T) over Y ⊆ S(T) and let r* be a nested relation over R(T). Then the *null extended total projection* of r* onto Y, $\Pi^{ne}\downarrow_Y(r^*)$, is a nested relation, over R(T′), defined recursively by:

(1) if T consists of a single node and ATT(ROOT(T)) = X ($\neq \Lambda$), then
$\Pi^{ne}{\downarrow}_Y(r^*) \cong \{t[Y] \mid t \in r^* \wedge DEF(t) = Y = S(T')\}$;

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

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

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

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

$DEF(t) = Y = S(T')\}$.

An immediate consequence of the above definition is that for all t ∈ $\Pi^{ne}{\downarrow}_Y$(r*), DEF(t) = Y = S(T′). Hence, a nested relation r*, over R(T), is said to be S(T)-total if and only if $\Pi^{ne}{\downarrow}_Y$(r*) = r*.

Next we give the definition of 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 A1.9. 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

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

(2) if R(T_i) = X_i(R(T^i_1))*(R(T^i_2))* ... (R($T^i_{s_i}$))*, i = 1,2, where X_i = ATT(ROOT(T_i)) and T^i_1, T^i_2, ..., $T^i_{s_i}$, s_i ≥ 1, denote the first level subtrees of T_i, then

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

(2.1) $(t[X_1X_2] \cong t_1[X_1] \bowtie^{ne} t_2[X_2] \wedge$

$t[X_1X_2] \neq \{null\}$ if $((X_1 \neq \Lambda) \vee (X_2 \neq \Lambda))) \wedge$

(2.2) (let j ∈ {1,2,...,s_1}, k ∈ {1,2,...,s_2}, then

(2.2.1) $\forall$ j, k such that $S(T^1_j) \cap S(T^2_k) \neq \emptyset$:

$(t[(R(T)[S(T^1_j) \cup S(T^2_k)])^*] \cong$

t_1[(R(T_j^1))*] $\bowtie^{ne}$ t_2[(R(T_k^2))*]) ≠ {*null*} ∨

(2.2.2) ∀ j such that ∀ k S(T_j^1) ∩ S(T_k^2) = ∅:

t[(R(T_j^1))*] ≅ t_1[(R(T_j^1))*] ∨

(2.2.3) ∀ k such that ∀ j S(T_j^1) ∩ S(T_k^2) = ∅:

t[(R(T_k^2))*] ≅ t_2[(R(T_k^2))*])}.

We define the *null extended Cartesian product* operator, denoted as $\times^{ne}$, as a special case of our null extended join operator. That is, the null extended join reduces to the null extended Cartesian product operator when S(T_1) ∩ S(T_2) = ∅.

The definition of the *null extended meet operator*, which computes the GLB of two nested relations, now follows.

DEFINITION A1.10. Let r_1 and r_2 be nested relations over a NRS, R(T). The *null extended meet* of r_1 and r_2, $r_1 \cap^{ne} r_2$, yielding r* over R(T), is defined recursively by:

(1) if T consists of a single node and ATT(ROOT(T)) = X, then

r* ≅ {t | ∃ t_1 ∈ r_1, ∃ t_2 ∈ r_2:

∀ A ∈ X, if t_1[A] ≅ t_2[A], then t[A] ≅ t_1[A], otherwise t[A] ≅ *null*;

(2) if R(T) = X(R(T_1))*(R(T_2))* ... (R(T_s))*, where X = ATT(ROOT(T)) and T_1, T_2, ..., T_s, s ≥ 1, denote the first level subtrees of T, then

r* ≅ {t | ∃ t_1 ∈ r_1, ∃ t_2 ∈ r_2: t[X] ≅ t_1[X] $\cap^{ne}$ t_2[X] ∧

(∀ i ∈ {1,2,...s} t[(R(T_i))*] ≅ t_1[(R(T_i))*] $\cap^{ne}$ t_2[(R(T_i))*])}.

The last definition allows us to pad tuples over a projected NRS, R(T′), of a NRS, say R(T), with *nulls*; the definition is needed when we consider tuples over a NRI containing *nulls*.

DEFINITION A1.11. Let r* be a nested relation over a projected NRS, R(T′), of a NRS, R(T), where S(T′) = Y. Then, where t is a tuple in r*,

PAD(t) is a tuple over the NRS, R(T), such that PAD(t)[Y] ≅ t ∧ (PAD(t)[S(T) − Y] ≅ *null*).

We can now naturally generalise PAD(t) to PAD(r*), where r* is a nested relation over a projected NRS, R(T′), of R(T), namely

PAD(r*) is a nested relation over the NRS, R(T), such that PAD(r*) ≅ {PAD(t) | t ∈ r*}.

Finally, we can apply PAD to a nested database, d* = {r_1^*, r_2^*, ..., r_q^*}, over a joinable NDS, **R**(F), over the joined NRS, U(T), namely

`PAD(d*) is a nested relation over the joined NRS, U(T), such that PAD(d*)` $\cong \cup^{ne}\ {}_{i=1}^{q}$ `(PAD(`r_i^*`)).`

APPENDIX 2. NULL EXTENDED JOIN DEPENDENCIES

Herein we define the NEJD, which incorporates the definition of the null extended join operator into the definition of the NJD. The NJD generalises the JD [5] to flat relations which may contain *nulls*.

We begin by employing the following useful notation for FDSs found in [5]. *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 **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.

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 NDS **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, **R**(F), if FDS(F′) ∈ SUBSET(FDS(F)). We denote the set of all covering subset NDSs of **R**(F) by SUBSET(**R**(F)).

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

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

where $S(T'_i) \in$ FDS(F′), 1 ≤ i ≤ n. Consequently, t*[MANY(FDS(F′))] is MANY(FDS(F′))-total, due to Definition A1.9 of $\bowtie^{ne}$. Hereafter, we abbreviate this to: *the tuples*, $t_1, t_2, ..., t_n \in r^*$, *are combinable with the resulting tuple*, *t**, *over U(T).*

We next define the NEJD by utilising the above concepts.

DEFINITION A2.1. 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.

Next we define the NEJD-rule for the NEJD, $\bowtie^{ne}$[**R**(F)], with respect to a nested relation, r*, over the joined NRS, U(T).

DEFINITION A2.2. Let **R**(F) be a joinable NDS over the joined NRS, U(T); let $\bowtie^{ne}$[**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, $\bowtie^{ne}$[**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 $t'^* \in$ r* such that $t^* \leq t'^*$, then $RULE_{\bowtie^{ne}[\mathbf{R}(F)]}(r^*) \cong r'^*$, where r′* is r* after the assignment $r'^* := r^* \cup^{ne} \{t^*\}$.

The next theorem shows the correspondence between the NEJD and the NJD.

THEOREM A2.1 [17]. *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)], holds in r* *if and only if the NJD,* $\bowtie$[*FDS(F)*], holds in $\mu^*(r^*)$. □

We note that we could generalise NEJDs to *embedded NEJDs* by defining an embedded NEJD to hold over a subset of the NURS, U(T), rather than the whole of U(T). In the special case of flat relations we investigated the inference problem of embedded NJDs (NJDs are a special case of embedded NJDs) in [21], wherein we showed that an extension of the chase procedure, called the *or-chase*, allows us to solve the inference problem for embedded NJDs.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous referee for his constructive comments.

REFERENCES

1. S. ABITEBOUL AND N. BIDOIT, Non first normal form relations: An algebra allowing data restructuring, *J. Comput. System Sci.* **33**, No. 3 (1986), 361-393.

2. P. ATZENI AND R. TORLONE, Updating databases in the weak instance approach, *in* "Proceedings of the 6th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems," Philadelphia, Penn., ACM, New York, 1989, pp. 101-109.

3. P. ATZENI AND M.C. BERNARDIS, A new interpretation for null values in the weak instance model, *J. Comput. System Sci.* **41**, No. 1 (1990), 25-43.

4. F. BANCILHON AND S. KHOSHAFIAN, A calculus for complex objects, *J. Comput. System Sci.* **38**, No. 2 (1989), 326-340.

5. C. BEERI AND M.Y. VARDI, 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.

6. N. BIDOIT, The Verso algebra or how to answer queries with fewer joins, *J. Comput. System Sci.* **35**, No. 3 (1987), 321-364.

7. P. BUNEMAN, A. JUNG, AND A. OHORI, Using powerdomains to generalize relational databases, *Theoret. Comput. Sci.* **91**, (1991), 23-55.

8. E.F. CODD, Extending the database relational model to capture more meaning, *ACM Trans. Database Systems* **4**, No. 4 (1979), 397-434.

9. B.A. DAVEY AND H.A. PRIESTLY, "Introduction to Lattices and Order," Cambridge University Press, Cambridge, U.K., 1990.

10. R. FAGIN, A.O. MENDELZON, AND J.D. ULLMAN, A simplified universal relation assumption and its properties, *ACM Trans. Database Systems* **7**, No. 3 (1982), 343-360.

11. R. FAGIN, Degrees of acyclicity for hypergraphs and relational database systems, *J. ACM* **30**, No. 3 (1983), 514-550.

12. M.H. GRAHAM AND M. YANNAKAKIS, Independent database schemas, *J. Comput. System Sci.* **28**, No. 1 (1984), 121-141.

13. M.H. GRAHAM, A.O. MENDELZON, AND M.Y. VARDI, Notions of dependency satisfaction, *J. ACM* **33**, No. 1 (1986), 105-129.

14. Y. GUREVICH AND S. SELAH, Fixed-point extensions of first-order logic, *Ann. Pure Appl. Logic* **32**, (1986), 265-280.

15. P. HONEYMAN, Testing satisfaction of functional dependencies, *J. ACM* **29**, No. 3 (1982), 668-677.

16. T. IMIELINSKI AND W. LIPSKI JR., Incomplete information in relational databases, *J. ACM* **31**, No. 4 (1984), 761-791.

17. M. LEVENE AND G. LOIZOU, Semantics for null extended nested relations, *ACM Trans. Database Systems* **18**, No. 3 (1993), 414-459.

18. M. LEVENE AND G. LOIZOU, A domain theoretic characterisation of the universal relation, *Internat. J. Comput. Math.* **40**, (1991), 69-74.

19. M. LEVENE AND G. LOIZOU, A fully precise null extended nested relational algebra, *Fundamenta Informaticae*, in press.

20. M. LEVENE, "The Nested Universal Relation Database Model," Lecture Notes in Computer Science Vol. 595, Springer-Verlag, Berlin, 1992.

21. M. LEVENE AND G. LOIZOU, Inferring null join dependencies in relational databases, *BIT* **32**, (1992), 413-429.

22. Y.E. LIEN, On the equivalence of database models. *J. ACM* **30**, No. 2 (1982), 333-362.

23. D. MAIER, A.O. MENDELZON, AND Y. SAGIV, Testing implication of data dependencies, *ACM Trans. Database Systems* **4**, No. 4 (1979), 455-469.

24. D. MAIER, J.D. ULLMAN, AND M.Y. VARDI, On the foundations of the universal relation model, *ACM Trans. Database Systems* **9**, No. 2 (1984), 283-308.

25. D. MAIER, D. ROZENSHTEIN, AND D.S. WARREN, Window functions, *in* "Advances in Computing Research," Vol. 3, (P.C. Kanellakis and F. Preparata, Eds.), JAI Press, Greenwich, 1986, pp. 213-246.

26. A.O. MENDELZON, Database states and their tableaux, *ACM Trans. Database Systems* **9**, No. 2 (1984), 264-282.

27. Y. SAGIV, A characterization of globally consistent databases and their correct access paths, *ACM Trans. Database Systems* **8**, No. 2 (1983), 266-286.

28. H.-J. SCHEK AND M.H. SCHOLL, The relational model with relation-valued attributes, *Inf. Systems* **11**, No. 2 (1986), 137-147.

29. D.A. SCHMIDT, "Denotational Semantics: A Methodology for Language Development," Allyn and Bacon, Inc., Newton, Ma., 1986.

30. J. STEIN AND D. MAIER, 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.

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

32. J.D. ULLMAN, Universal relation interfaces for database systems, *in* "Proceedings of 9th IFIP World Computer Congress," Paris, 1983, pp. 243-252.

33. J.D. ULLMAN, "Principles of Database and Knowledge-Base Systems," Vol. 1, Computer Science Press, Rockville, Md., 1988.

34. D. VAN GUCHT AND P.C. FISCHER, Multilevel nested relational structures, *J. Comput. System Sci.* **36**, No. 1 (1988), 77-105.

35. M.Y. VARDI, The universal-relation data model for logical independence, *IEEE Software* **5**, (1988), 80-85.